TECHNICKÁ UNIVERZITA V LIBERCI

Fakulta mechatroniky a mezioborvých inženýrských studií

DIPLOMOVÁ PRÁCE

Prostředek pro ověření úplnosti testu

Tool for Test Integrity Check

Liberec 2005 Jaroslav Mandík

TECHNICKÁ UNIVERZITA V LIBERCI

Fakulta mechatroniky a mezioborových inženýrských studií

Katedra elektroniky a zpracování signálů (KES) Akademický rok: 2004/2005

ZADÁNÍ DIPLOMOVÉ PRÁCE

pro: **Jaroslava MANDÍKA**

studijní program: M 2612 – Elektrotechnika a informatika

obor: 3902T005 – Automatické řízení a inženýrská informatika

Vedoucí katedry Vám ve smyslu zákona o vysokých školách č.111/1998 Sb. určuje tuto diplomovou práci:

Název tématu:

Prostředek pro ověření úplnosti testu

Zásady pro vypracování:

1. Seznamte se obvodem ispLSI1016EA a s jeho programovacím rozhraním.
2. Seznamte se s jazykem VHDL pro popis elektronických systémů a s využitím VHDL jazyka popište vzorové obvody a navrhněte řešení popisu poruch.
3. Sestavte program, který pomocí grafického rozhraní do libovolného HDL popisu doplní libovolnou poruchu (stack-at0, stack-at1, short) pro uživatelem vybraný signál. Vytvořte databázi takovýchto souborů pro vybraný obvod.
4. Pomocí Delphi sestavte program pro komunikaci mezi PC(LPT) a ispLSI obvodem. Program zajistí komunikaci s obvodem, vizualizaci stavu úlohy a zobrazení výsledků.

5. Sestavte program, který bude komunikovat s vybraným obvodem, nastavovat vstupy obvodu, sledovat výstupy a zobrazuje je na obrazovce (volitelně i vnitřní stavy).

Rozsah grafických prací: dle potřeby dokumentace

Rozsah průvodní zprávy: cca 40 až 50 stran

Seznam odborné literatury:

[1] Skalický R., Vývojová deska pro diagnostické účely, DP TUL 2003.

[4] Kolouch, J.: Programovatelné logické obvody a modelování číslicových systémů v jazycích ABEL a VHDL. Skriptum FEI VUT Brno, 2000, ISBN 80-214-1733-1

[5] Dokumentace k obvodu ispLattice1016EA.

Vedoucí diplomové práce: Ing. Zdeněk Plíva, PhD.

Konzultant: Ing. Milan Kolář, CSc.

Zadání diplomové práce: **22. 10. 2004**

Termín odevzdání diplomové práce: **20. 5. 2005**

L.S.

....

Vedoucí katedry Děkan

V Liberci dne 22.10.2004

Prostředek pro ověření úplnosti testu

Anotace:

Cílem této diplomové práce je vytvořit model kombinačního obvodu s využitím VHDL jazyka, navrhnout řešení popisu poruch, sestavit program, který do VHDL popisu doplní libovolnou poruchu a vyvinout počítačovou aplikaci schopnou komunikovat s reálným obvodem po LPT a zobrazit průběh úlohy a výsledky.

Výsledkem práce je databáze modelů obvodu C17 obsahující různé poruchy a aplikace C17 Bugger umožňující uživateli úpravu VHDL popisu obvodu přidáním libovolné poruchy a C17 Simulator provádějící simulaci obvodu C17 a komunikaci s modelem obvodu nahraném v reálném programovatelném hradlovém poli ispLSI2032 od firmy Lattice Semiconductor.

Přínosem práce je možnost využití vytvořených aplikací při výuce studentů v předmětech zabývajících se číslicovými obvody nebo testováním integrovaných obvodů.

Tool for Test Integrity Check

Abstract:

The aim of the diploma thesis is to create a model of a combinatorial circuit using VHDL language, propose solution to describe failures, built a computer programme, which puts any failure in the VHDL code and design computer aplication, which is able to communicate with a real circuit via LPT and display the running process and the outcomes.

The result of this work is a database of the models of C17 circuits that contain various failures, computer aplications C17 Bugger enabling the modification of a VHDL code by adding a failure and C17 Simulator providing the simulation of circuit C17 and communication with the model uploaded in a real programmable gain array ispLSI2032 produced by Lattice Semiconductors.

The benefit of the work is the possibility of usage of the comoputer aplications in education of the students in subjects, which are engaged in digital circuits or integrated circuits testing.

Prohlášení

Byl jsem seznámen s tím, že na mou diplomovou práci se plně vztahuje zákon č. 121/2000 Sb. o právu autorském, zejména § 60 – školní dílo.

Beru na vědomí, že Technická univerzita v Liberci (TUL) nezasahuje do mých autorských práv užitím mé diplomové práce pro vnitřní potřebu TUL.

Užiji-li diplomovou práci nebo poskytnu-li licenci k jejímu využití, jsem si vědom povinnosti informovat o této skutečnosti TUL; v tomto případě má TUL právo ode mne požadovat úhradu nákladů, které vynaložila na vytvoření díla, až do jejich skutečné výše.

Diplomovou práci jsem vypracoval samostatně s použitím uvedené literatury a na základě konzultací s vedoucím diplomové práce a konzultantem.

V Liberci dne

Jaroslav Mandík

Poděkování

Na tomto místě bych rád poděkoval panu Ing. Zdeňku Plívovi, PhD. za vedení diplomové práce, cenné rady, poskytnuté informace, propůjčený hardware a velkou dávku trpělivosti.

Dále chci poděkovat panu Ing. Milanu Kolářovi, CSc. za cenné rady, trefné připomínky a poskytnuté informace.

Děkuji paní RNDr. Kláře Císařové za podporu a rady při úpravě textu diplomové práce.

Také děkuji panu Ing. Petru Novákovi, Dr. Ralfu Leiterovi a celkově firmě TRW Automotive za poskytnuté vybavení a vstřícný přístup při dokončování diplomové práce.

Obsah:

1. Úvod

Prvním z cílů diplomové práce je vytvořit model číslicového obvodu pomocí jazyka VHDL, konkrétně byl vybrán kombinační obvod C17. K tomu účelu bude nutné se seznámit s jazykem VHDL a s používáním návrhového systému ispDesignEXPERT od firmy Lattice Semiconductor. Návrh se zkompiluje a bude nahrán do programovatelného hradlového pole (FPGA) ispLSI2032 také od firmy Lattice Semiconductors.

Dále bude nutné nalezení řešení popisu poruch typu Stuck_at0, Stuck_at1 a ShortCut a způsob jejich zakomponování do originálního VHDL popisu. Na tento úkol bude přímo navazovat program, který bude úpravu VHDL souboru vložením poruchy provádět automaticky a bude uživateli nabízet vhodné prostředky pro výběr signálů, na kterých se má porucha vyskytnout.

Posledním cílem, který bude prezentovat výsledky dosažené v předchozích zmíněných bodech, bude aplikace umožňující komunikaci s modelem obvodu nahraném v reálném hradlovém poli ve smyslu posílání bitů na vstupy a čtení bitů z jeho výstupů. Tato aplikace by měla vhodným způsobem, nejlépe graficky, zobrazovat průběh práce s obvodem, nacházet předpokládané logické úrovně výstupů obvodu, případně určovat vnitřní stavy.

Obě výsledné aplikace budou vyvíjeny pro operační systém Windows a měli by být navrženy s ohledem na možné využití při výuce studentů v předmětech zabývajících se danou tématikou. To bude předpokládat uživatelsky přátelské prostředí aplikací a zároveň jejich robustnost a možnost pracovat na různých verzích OS Windows.

2. Teorie poruch v IO

2.1. Základní pojmy

V integrovaných obvodech, stejně jako v jakýchkoli jiných systémech, dochází k tomu, že se v nich z různých příčin mohou vyskytovat poruchy. Základní pojmy se používají tyto:

- **defekt** – Pokud se na IO podíváme z úrovně fyzikální, je defekt fyzická nedokonalost v obvodu zapříčiněná nesprávným návrhem, výrobou nebo jejich kombinací, případně dalšími výrobními procesy, které způsobují odchylku od předepsaných specifikací.
- **porucha –** O poruše mluvíme v souvislosti s logickou úrovní obvodu a je definována jako jev spočívající v ukončení schopnosti plnit požadovanou funkci podle technických podmínek
- **chyba –** Je nesprávná odezva v chování obvodu – projev poruchy nebo defektu
- **charakteristika poruch:**
  - **latentní porucha** – porucha, která je skrytá a nedá se detekovat, což znamená, že se neprojevuje chybou
  - **úplná porucha** – porucha, která vede k odlišné logické funkci obvodu
  - **částečná porucha** – změna fyzikálních parametrů, která ještě nemění logickou funkci, ale už přesahuje hranice stanovených technických podmínek
  - **jednotlivá porucha** – výskyt právě jedné poruchy v daném časovém okamžiku
  - **násobná porucha** – výskyt více jak jedné poruchy v daném časovém okamžiku
  - **ekvivalentní poruchy –** poruchy, které se projevují stejně, a není možné je rozlišit
- **model poruch** – definuje logiku anebo chování obvodu s poruchou

2.2. Typy poruch

Základní rozdělení poruch vzhledem k jejich výskytu je následující:

- stále poruchy – jsou v obvodu trvale a vznikají například chybou při výrobě
- nestálé poruchy – vyskytují se buďto přechodně nebo náhodně

Základní kategorie poruch:

- trvalé poruchy (stuck_at0, stuck_at1)
- přemostění a zkraty
- přerušení neboli trvale otevřeno – chybí vodivé propojení, časté u CMOS obvodů, kde je přerušen gate pull-up nebo gate pull-down tranzistoru
- poruchy citlivé na vzorek

- parametrické poruchy

Výskyt poruch může být (viz. kapitola 2.1)**:**

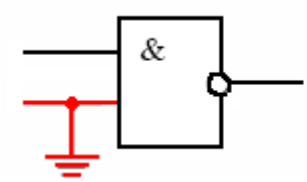
- jednotlivý
- násobný

2.3. Poruchy v mé diplomové práci

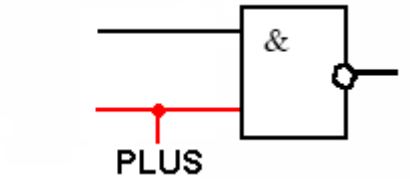
V mé diplomové práci jsem se zabýval poruchami, které jsou vzhledem k výskytu **jednotlivé** a **stálé**. Z několika druhů byly pak vybrány **trvalé poruchy** (stuck_at0, stuck_at1) a **zkraty**. Následuje rozbor uvedených poruch.

2.3.1. Model trvalých poruch

Model trvalých poruch vychází z průmyslového standardu z roku 1959. Předpokládá se defekt, který zapříčiní, že na jeho vstupu nebo výstupu je trvale držená hodnota logická 0 nebo logická 1.



Stuck_at 0 jako trvalý zkrat na zem



Stuck_at 1 jako trvalý zkrat na plus

Pravdivostní tabulka je v tomto případě zcela jednoduchá, ale pro úplnost je uvedena:

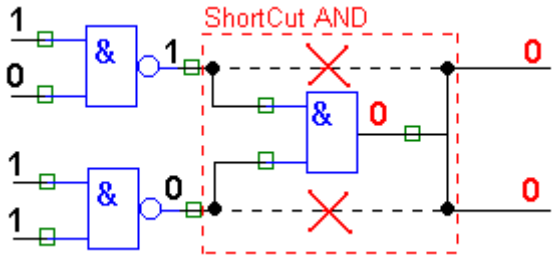
Signál	Stuck_at 0	Stuck_at 1
0	0	1
1	0	1

Tabulka 1: Pravdivostní tabulka Stuck_at0 a Stuck_at1

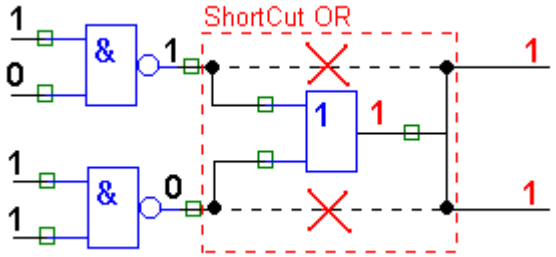
Model trvalých poruch se s úspěchem používal pro obvody technologie TTL, není však vhodný pro obvody typu CMOS.

2.3.2. Zkraty

Druhým druhem poruch, kterým jsem v diplomové práci zpracoval jsou zkraty. Ty jsou dominantními poruchami v současných CMOS obvodech. Jsou způsobeny prachem uloženým na čipu. Tyto poruchy v číslicových obvodech ovšem není možné brát jako prosté propojení dvou (nebo více) signálů. Při vyšetřování logického chování číslicového obvodu obsahujícího zkrat se uvažuje propojení signálu hradly typu „AND“ nebo „OR“. Tyto dva typy zkratů vysvětlují následující obrázky a pravdivostní tabulky:



Obrázek 1: Zkrat typu NAND



Obrázek 2: Zkrat typu OR

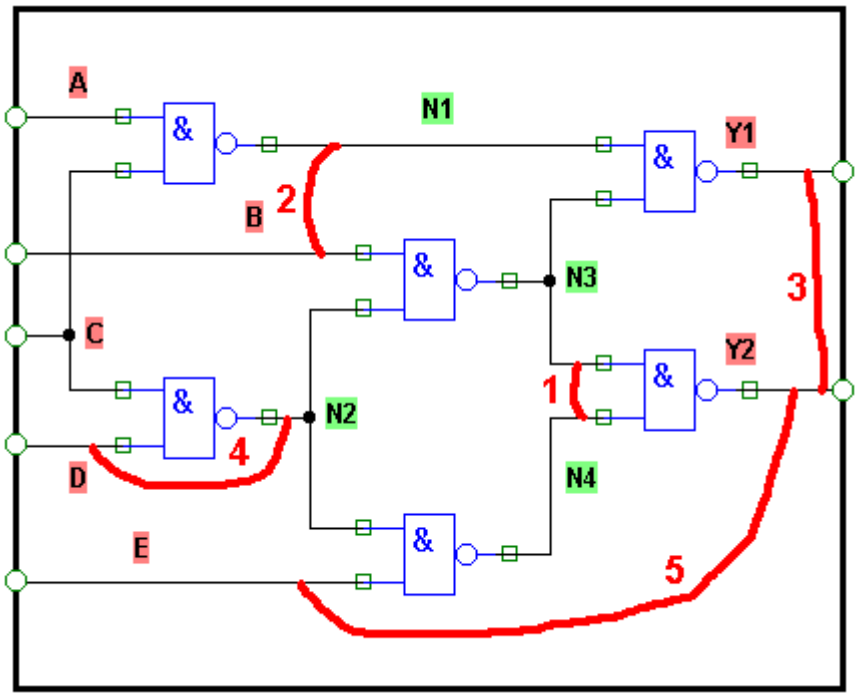
Z pravdivostní tabulky je vidět, že zkraty je možné reprezentovat běžnými hradly:

x	y	x ShortCut (AND) y
0	0	0
0	1	0
1	0	0
1	1	1

x	y	x ShortCut (OR) y
0	0	0
0	1	1
1	0	1
1	1	1

Tabulka 2: Pravdivostní tabulka zkratů AND a OR

Dále se poruchy typu zkratů rozdělují podle toho, které vodiče v obvodu zkratují. Z tohoto úhlu pohledu je možné rozlišit zkraty, které vytvářejí nebo naopak nevytvářejí zpětnou vazbu.



Obrázek 3: Několik příkladů zkratů v obvodu C17

Na obrázku je znázorněno několik příkladů zkratů, které mohou nastat v obvodu C17:

1. n3 / ShortCut n2 - zkratovány vstupní signály jednoho hradla NAND – **nevzniká ZV**
2. n1 / ShortCut b - zkratovány dva zcela nesouvisející signály – **nevzniká ZV**
3. y1 / ShortCut y2 - zkratovány výstupní signály – **nevzniká ZV**
4. n2 / ShortCut d – zkratován výstup hradla s jedním z jeho vstupů – **vzniká ZV**
5. y2 / ShortCut e – signál **e**, ač přes několik hradel, ovlivňuje signál **y2** – **vzniká ZV**

Zkraty **1**, **2** a **3** se nazývají kombinační, protože nemění kombinační charakter obvodu. Naopak zpětnovazební zkraty mění obvod na sekvenční, který není možné popsat pravdivostní tabulkou. Vzájemným propojením výstupů a vstupů dvou hradel NAND vzniká například RS klopný obvod. Oba druhy zkratů není složité za použití strukturálního popisu popsat jazykem VHDL. Ovšem vyřešit takový obvod, zvláště pokud se zkrat „vrací“ přes více než jedno hradlo je poměrně komplikovanou záležitostí.

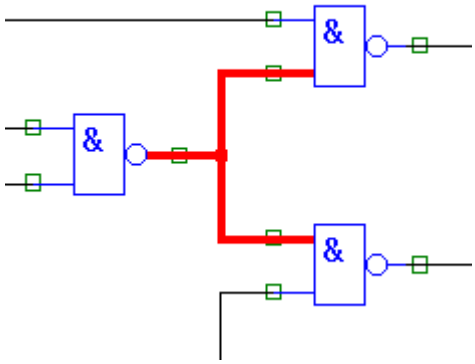
Dalšími možnostmi, jak je možné zkraty rozlišovat jsou zkraty nízkoodporové, vysokoodporové, dominantní, inter-gate a intra-gate. Těmito klasifikacemi se diplomová práce nezabývá.

2.3.3. Šíření poruch ve větvení obvodu

Pokud se pracuje s vkládáním poruch do obvodu, který obsahuje větvení, musí se zodpovědět dvě důležité otázky.

1. Implikuje porucha na kmeni větvení také poruchu na větvích?
2. Implikuje porucha na některé větvi poruchu na kmeni, resp. celém stromu?

Já jsem si na obě otázky odpověděl **ANO**. Všechny signály jsem uvažoval víceméně abstraktně. Neřešil jsem, kde se v obvodu vyskytují nebo jestli větvení má nějaký vliv na jejich šíření. Nebral jsem tedy v úvahu ani to, zda se porucha u větvícího se signálu týká větve nebo kmene.



Obrázek 4: Šíření chyby ve větvení obvodu

3. Použité prostředky

3.1. Vývojové prostředí pro aplikaci – Delphi 5

Pro vývoj aplikací C17 SIMULATOR a C17 Bugger jsem použil ověřený návrhářský software Delphi 5. Tento software umožňuje vysoce efektivní tvorbu aplikací pro Windows. Je založený na formulářích (windowsovksých oknech) a objektově orientovaných vlastnostech programovacího jazyka Object Pascal. Grafickou podobu zmíněných oken může designér uživatelsky příjemně navrhovat a používat při tom množství ovládacích a zobrazovacích prvků z knihovny vizuálních komponent (VCL). Události, vzniklé používáním komponent uživatelem, jsou ošetřeny kódem psaným jazykem Object Pascal, který je hlavním pilířem tohoto vývojového prostředí.

V Delphi je možné během poměrně krátké doby vytvořit jednoduchý, fungující program. Chci ovšem na tomto místě upozornit na skutečnost, že detailně vyladit aplikaci jak po grafické, tak hlavně po funkční a tedy programové stránce, dá téměř vždy mnohem více práce, než je na první pohled vidět a vynaložené úsilí není často doceněno.

3.1.1. Zpracování výjimek

Smyslem výjimek je zvýšit odolnost programů jejich doplněním o schopnost jednoduchého a jednotného řešení softwarových i hardwarových chyb. Program může takové chyby přečkat nebo elegantně skončit a před tím dát uživateli možnost uložit si rozpracovaná data. Mechanismus výjimek umožňuje oddělit zpracování chyb od normálního kódu, namísto jejich vzájemného provázání. Výsledný kód je pak kompaktnější a méně promíchaný s údržbou, která se nevztahuje k vlastním cílům programování.

3.1.2. ??? něco dalšího ???

3.2. VHDL

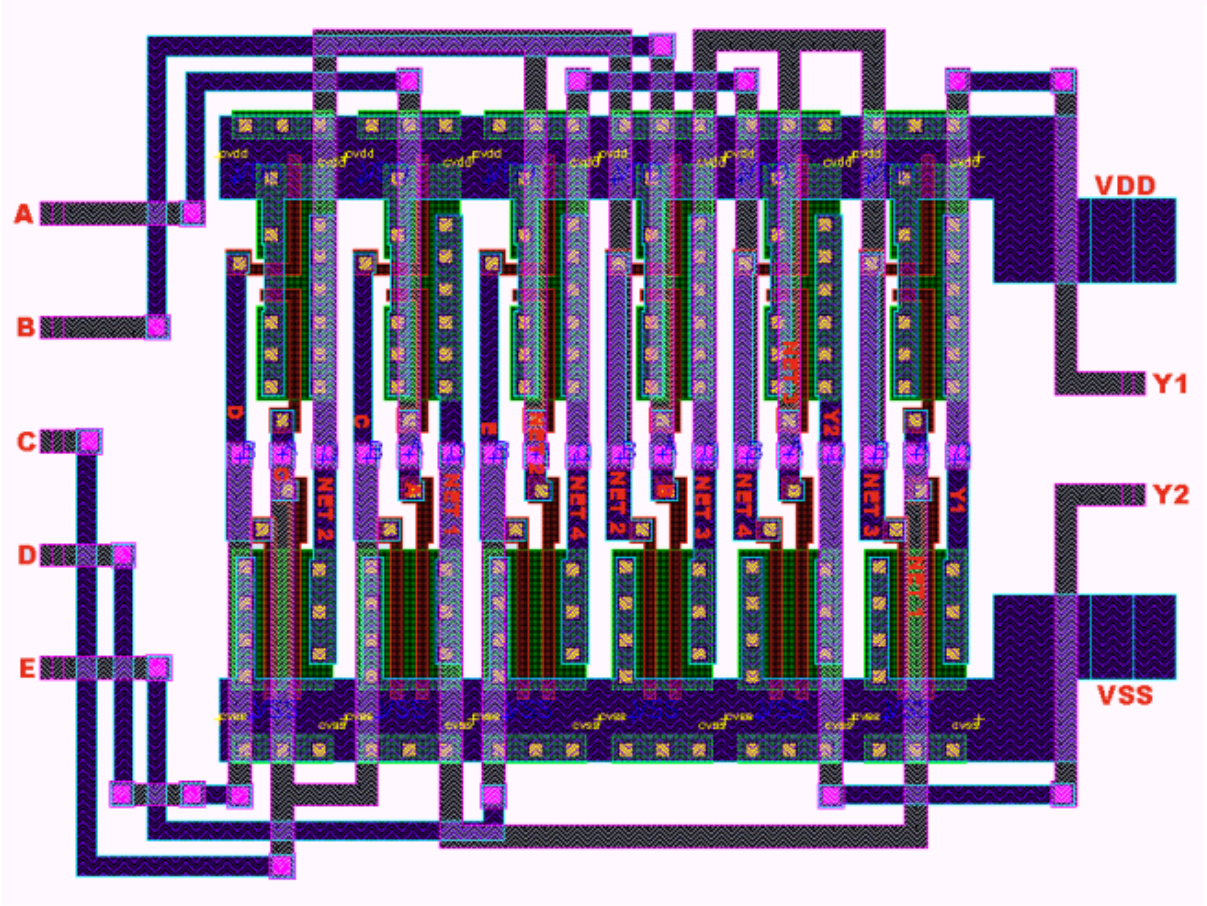
K popisu obvodu C17 byl vybrán jazyk **VHDL** (**V**ery High Speed Integrated Circuits **HDL**). Je to jazyk vyvinutý a používaný pro popis technických prostředků elektronických systémů. Je vhodný pro návrh metodou shora-dolů a vzniklý model není závislý na budoucí technologii realizace. VHDL patří mezi paralelní jazyky a umožňuje tak popis více časově souběžných procesů.

K editaci kódu, simulacím, kompilaci a nahrání obvodu do FPGA byl použit software **ispDesignEXPERT** od firmy Lattice Semiconductor. Výchozím nástrojem, který jsem

k vytváření modelu využil je Project Navigator. Ten umožňuje založení projektu reprezentujícího daný návrh, popsání procesů pomocí editoru zdrojového kódu, nalezení syntaktických i logických chyb, kompilaci projektu, časovou simulaci a vytvoření JEDEC souboru, který je možné využít k implementaci do cílového zařízení.

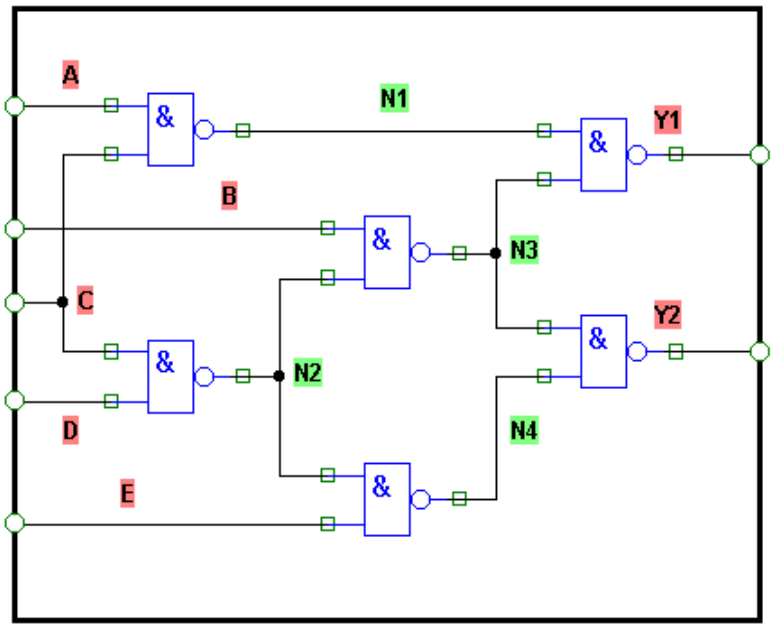
3.3. Obvod C17

Číslicový kombinační obvod C17 patří do řady zkoušecích obvodů. Používají se při testování a ladění jiných systémů, např. tak, jak je využit v mě diplomové práci. Obvod nemá v podstatě žádnou konkrétní funkci, má jednoduchou a jasnou strukturu a provádí nepříliš složitou kombinační logiku.



Obrázek 5: Pohled na křemík obvodu C17

Obvod lze popsat následující pravdivostní tabulkou, která je uvedená v Příloze B. Obvod je složen ze šesti hradel NAND. Schéma jejich propojení vypadá takto:



Obrázek 6: Schéma obvodu C17

3.4. FPGA

??? Pár informací o programovatelném hradlovém poli.

3.5. Propojení PC s hradlovým polem ispLSI2032

Propojení PC, resp. aplikace C17 Simulator a reálného obvodu (modelu obvodu C17 nahraném v FPGA Lattice) je realizováno pomocí paralelního portu.

3.5.1. Stručná historie LPT

Paralelní port vznikl v roce 1981 jako alternativa k pomalému sériovému portu. Byl určen především k připojení tiskáren. Později, v roce 1991 NPA definovala novou množinu parametrů, které by zajistily snadné ovládání tiskáren po síti. Vznikem potřeby inovovat paralelní port na výkonnější a obousměrný, ale zároveň kompatibilní se stávajícím, vyzvala v roce 1994 NPA konsorcium IEEE, aby vytvořilo nový standart – normu IEEE 1284. To umožnilo obousměrný přenos dat rychlostí až 1 MB/s. Počet vstupů a výstupů se nezměnil, změnil se jen jejich význam a nově byl zaveden hardwarový handshaking (jednoduchý komunikační protokol).

3.5.2. Popis LPT

Ze softwarového hlediska není LPT nic víc než tři osmibitové registry. Obvykle jsou v PC dostupné tři paralelní porty, a to paralelní port na videokartě, LPT1 a LPT2.Jejich registry jsou umístěny na adresách:

- 3BCh-3BFh - paralelní port na videokartě
- 378h-37Fh - LPT1
- 278h-27Fh - LPT2

Adresa první registru každého portu se označuje jako bázová. Na této adrese se nachází **registr datového portu**. Je to 8 výstupních bitů určených k posílání dat. U základní verze LPT je tento registr pouze výstupní.

Bit	Jméno	Pin
0	D0	2
1	D1	3
2	D2	4
3	D3	5
4	D4	6
5	D5	7
6	D6	8
7	D7	9

Tabulka 3: Registr datového portu LPT

Na adrese (bázová adresa + 1) je **registr stavového portu**, který je určen výhradně pro čtení. V případě připojení tiskárny jsou tyto bity používány k určení jejího stavu. Jinak je možné je využít jako vstupní bity pro libovolná data. Je však nutné pro tento účel vytvořit vlastní programovou obsluhu. Registr vypadá následovně:

Bit	Jméno	Pin
0	rezervován	nezap.
1	rezervován	nezap.
2	notIRQ	nezap.
3	Error	15
4	Select In	13
5	Paper Out	12

6	Ack	10
7	/ Busy	11

Tabulka 4: Registr stavového portu LPT

Pozn.: Bit 7 je invertující, což je označeno znakem „/“. Je to použito i u následujícího registru.

Posledním registrem, který se nachází na adrese (bázová adresa + 2) je **registr řídícího portu**. Ten je originálně určen k řízení tiskárny a konfiguraci rozhraní. Má čtyři výstupní bity, které jsou ovšem většinou zapojeny jako otevřený kolektor se zdvihacím odporem a je možné je i číst. Vzhledem ke kompatibilitě je však vhodné je využívat jen pro čtení. Opět je tyto bity, s příslušnou programovou obsluhou, možné použít dle vlastního uvážení.

Bit	**Jméno**	**Pin**
0	/ Strobe	1
1	/ Auto LF	14
2	Initialize	16
3	/ Select Printer	17
4	Enable IRQ	nezap.
5	Enable Bi-direct	nezap.
6	nepoužit	nezap.
7	nepoužit	nezap.

Tabulka 5: Registr stavového portu LPT

Pokud se podíváme na paralelní port jako na hardware, je to zásuvka 25ti pinového konektoru typu D-sub. Nachází se zde dohromady 12 výstupních pinů (z nichž 8 patří datovému portu a 4 portu řídícímu) a 4 vstupní piny stavového registru. Na zbylých pinech konektoru D-sub, to jsou piny18 až 25, je připojena zem. U některých portů nemusí být všechny tyto piny propojeny, a tak je lepší je pro jistotu všechny spojit. Pokud se toto neprovede, výstupní komunikace většinou funguje, ale nelze číst data ze vstupních pinů. LPT je většinou součástí základní desky PC, někdy ho je však možné najít i na samostatné kartě. Rozdíl je zde jen ten, že pokud dojde, nejčastěji vlivem špatného zacházení, např. zkratováním pinů výstupního portu k zemi, k poškození portu, pak v případě integrovaného LPT na základní desce, to může vést k zničení i dalších jejích komponent.

3.5.3. Režimy LPT

Paralelní port může pracovat v některém z těchto režimů:

- **Standart Paralle Port** – SPP neboli normální paralelní port. V tomto režimu funguje LPT tak, jak je popsán výše a zejména je důležité to, že bity D0 – D7 datového portu fungují pouze jako výstupní
- **Bi-directional Parallel Port** – obousměrný SPP, někdy označovaný také jako SP/2. V tomto módu je možné využít registr datového portu obousměrně.
- **Enahanced Parallel Port** – EPP neboli vylepšený paralelní port. V tomto případě lze provádět 16-32bitovou komunikaci.
- **Enhanced Capabilities Parallel Port** – ECP neboli paralelní port s rozšířenými schopnostmi. Tento režim umožňuje emulaci všech předešlých módů.

Všechny počítače s procesory 386 a 486 pracují v režimu SPP, případně Bi-directional (PS/2). Stejně tak i přídavné karty do sběrnice ISA. To je však v dnešní době už poněkud historie. Moderní PCčka podporují i poslední dva zmíněné režimy. Módy EPP a ECP jsou rozšířené o hardwarový handshaking a jejich použití umožňuje vyšší přenosové rychlosti. Vyžaduje ale komplikovanější obsluhu na straně připojené periferie. Pro běžné aplikace, např. jako v této diplomové práci, je proto není nutné používat, především kvůli tomu, že by to mohlo způsobovat zbytečné komplikace.

3.5.4. Programová obsluha LPT

Pro softwarovou obsluhu jsem v mé DP použil unitu ParaPort.pas, která ??? napsat, kdo jí vytvořil a jestli byla součástí nějaký diplomky??? případně co umí ???

4. Vyvinutý software

4.1. VHDL popis obvodu C17

Existuje několik možností, jak pomocí VHDL popsat elektronický systém. **Behaviorální styl**, popisující obvod s vysokou úrovní abstrakce – využívá se především pro simulaci, **styl popisující tok dat**, vhodný zejména pro syntézu, a **strukturální styl**, kde se systém sestavuje z předpřipravených komponent a je možné provádět detailní časové simulace. Pro popis obvodu C17 jsem použil poslední zmíněný postup.

4.1.1. Použité komponenty

Samotný obvod C17 obsahuje jeden druh logického hradla – **NAND**, které je popsáno následujícím kódem:

```
ENTITY hr_nand IS
  PORT (in1, in2 : IN std_logic;
             out1 : OUT std_logic);
END ENTITY;

ARCHITECTURE behavior OF hr_nand  IS
BEGIN
  PROCESS (in1, in2)
  BEGIN
    out1 <=  NOT(in1 AND in2);
  END PROCESS;
END behavior;
```

Klíčové slovo ENTITY definuje logické vstupy a výstup hradla, ARCHITECTURE popisuje chování této komponenty. Zpoždění hradel se běžně nenastavuje, protože je implicitně přepsáno hodnotami vhodnými pro použité hradlové pole.

Z kapitoly 3 dále vyplývá, že pro vkládání poruchy ShortCut je potřeba hradlo **AND** a **OR**. Způsob popisu je identický s hradlem NAND:

```
ENTITY hr_and IS
  PORT (in1, in2 : IN std_logic;
             out1 : OUT std_logic);
END ENTITY;

ARCHITECTURE behavior OF hr_and  IS
BEGIN
  PROCESS (in1, in2)
```

```
  BEGIN
    out1 <= (in1 AND in2);
  END PROCESS;
END behavior;
--------------------------------------
ENTITY hr_or IS
  PORT (in1, in2 : IN std_logic;
             out1 : OUT std_logic);
END ENTITY;

ARCHITECTURE behavior OF hr_or  IS
BEGIN
  PROCESS (in1, in2)
  BEGIN
    out1 <= (in1 OR in2);
  END PROCESS;
END behavior;
```

4.1.2. Popis struktury C17

Popis obvodu C17 je realizován takto:

```
library ieee;
use ieee.std_logic_1164.all;

ENTITY c17 IS
PORT(a, b, c, d, e: IN std_logic;
        y1, y2: OUT std_logic);
END ENTITY;

ARCHITECTURE struct OF c17 IS
CONSTANT Stuck_at1: std_logic := '1';
CONSTANT Stuck_at 0: std_logic := '0';
SIGNAL n1, n2, n3, n4: std_logic;
SIGNAL ch , stoka, xy1, xy2: std_logic;
COMPONENT hr_nand
  PORT (in1, in2: IN std_logic;
             out1: OUT std_logic);
END COMPONENT;
COMPONENT hr_and
  PORT (in1, in2: IN std_logic;
             out1: OUT std_logic);
END COMPONENT;
COMPONENT hr_or
  PORT (in1, in2: IN std_logic;
             out1: OUT std_logic);
END COMPONENT;
BEGIN
  u0: hr_nand PORT MAP (in1=>a, in2=>c, out1=>n1);
  u1: hr_nand PORT MAP (in1=>c, in2=>d, out1=>n2);
```

```
  u2: hr_nand PORT MAP (in1=>b, in2=>n2, out1=>n3);
  u3: hr_nand PORT MAP (in1=>n2, in2=>e, out1=>n4);
  u4: hr_nand PORT MAP (in1=>n1, in2=>n3, out1=>y1);
  u5: hr_nand PORT MAP (in1=>n3, in2=>n4, out1=>y2);
END struct;
```

ENTITY c17 má 5 logických vstupů a 2 výstupy. Konstanty (CONSTANT) **Stuck_at1** a **Stuck_at0** jsou používány pro vkládání poruchy typu Stuck_at 0 a Stuck_at 1. Signály (SIGNAL) **n1**, **n2**, **n3** a **n4** reprezentují vnitřní stavy obvodu. Signálů **ch**, **stoka**, **xy1** a **xy2** se využije v případě vložení poruchy ShortCut.. Dále následují odkazy na výše zmíněné komponenty. V definiční části pak je popsána struktura obvodu tak, jak je možné ji vidět na Obrázku 6.

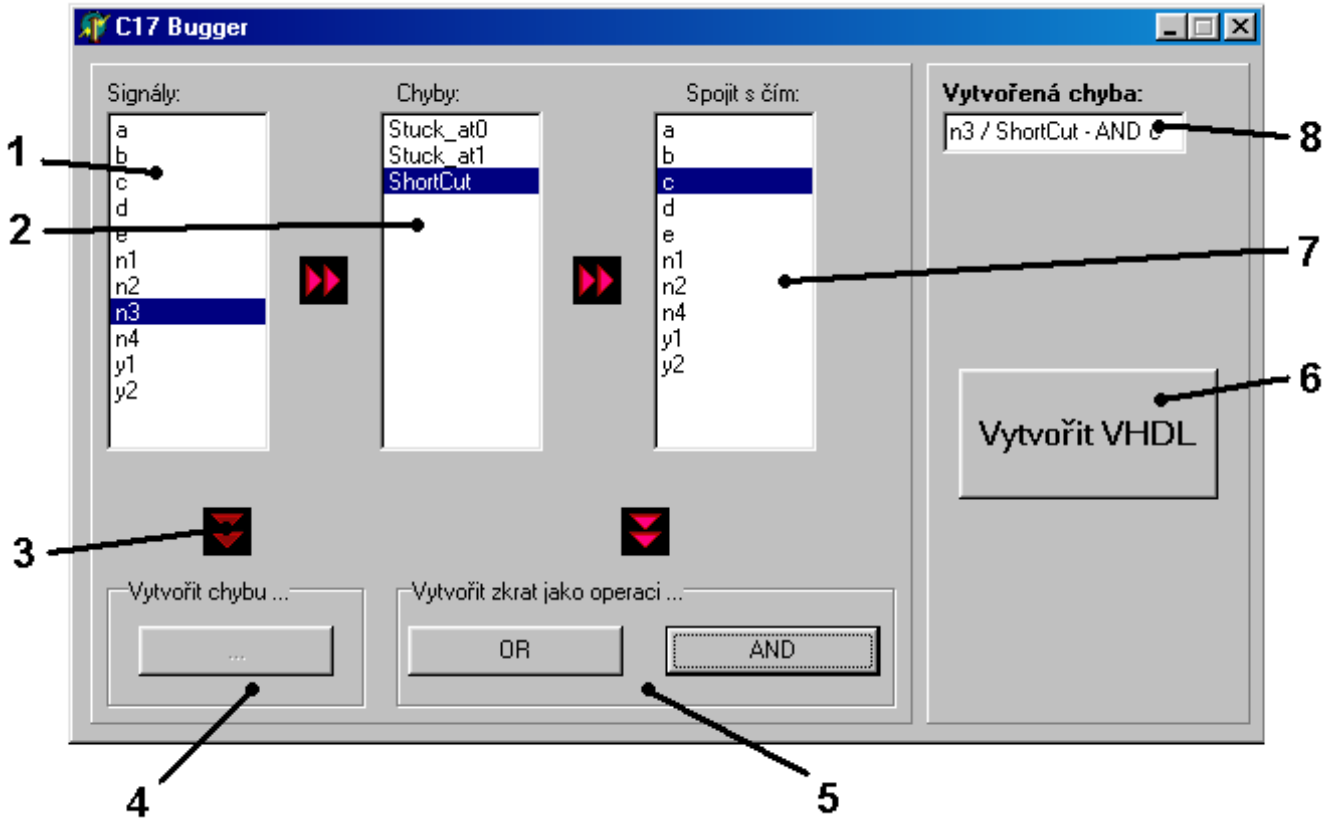
4.1.3. Úpravy VHDL popisu

Úpravou VHDL popisu obvodu C17 je myšlena změna původního VHDL souboru a jeho doplnění tak, že obsahuje právě jednu poruchu z kapitoly 2. Vkládání poruchy a vytváření popisů takto změněných obvodů se provádí aplikací C17 Bugger, jejíž funkce je vysvětlena hned v následující kapitole. Smyslem není ani tolik vysvětlovat popsání daných poruch pomocí jazyka VHDL, protože způsobů a přístupů je možné nalézt mnoho. Podstatou je možnost, na používání jednoduchým nástrojem, vložit do obvodu poruchu a následně, pomocí aplikace C17 Simulator pozorovat, jak se tato porucha projevuje chybou na výstupech a porovnávat toto chování s obvodem bez poruch.

4.2. Aplikace C17 Bugger

4.2.1. Manuál

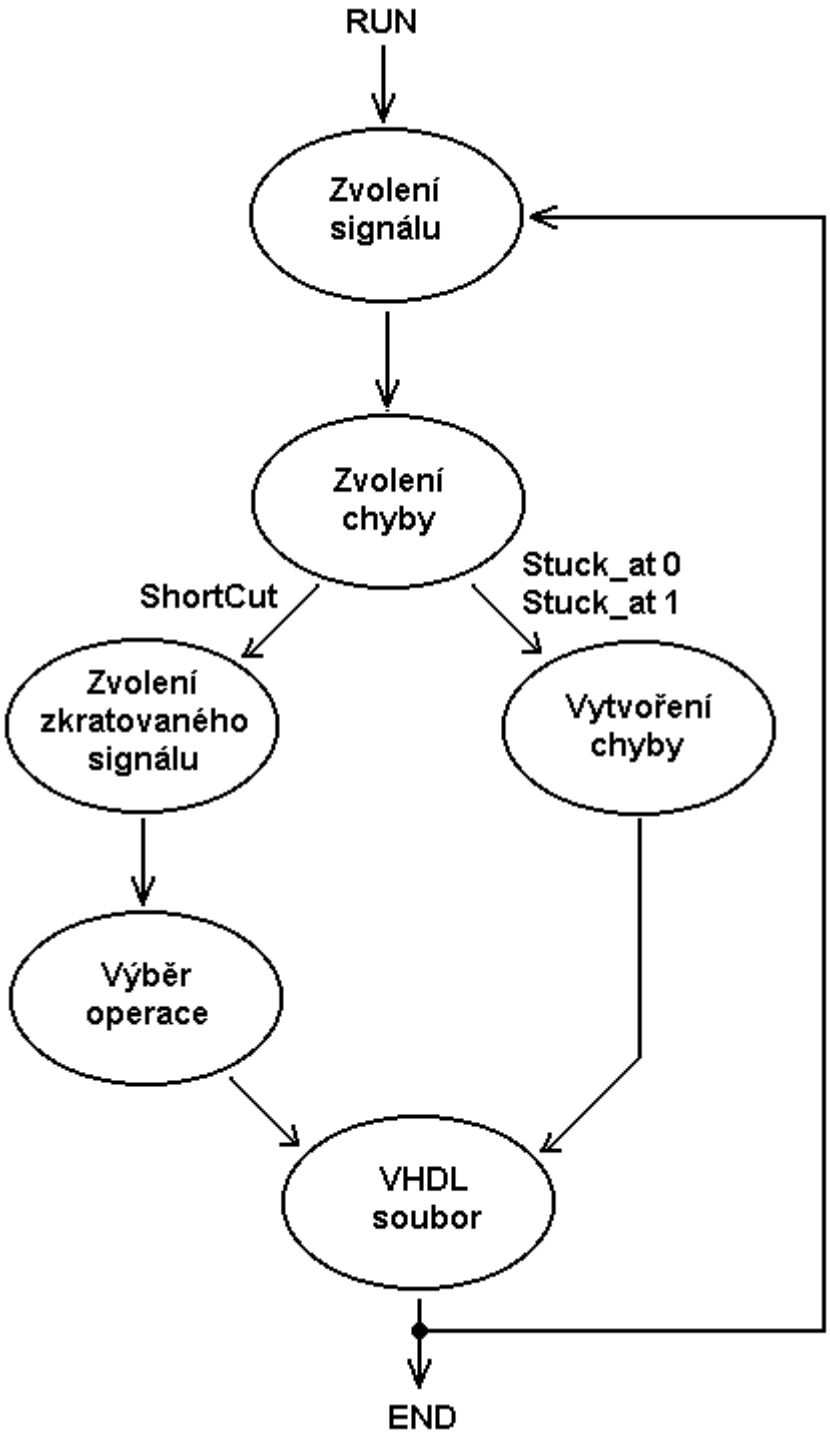
Aplikace C17 Bugger slouží vytvoření a následnému zakomponování poruchy do VHDL popisu obvodu C17.



Obrázek 7: Okno aplikace C17 Bugger

1. Seznam signálů
2. Seznam poruch
3. Navigační šipky vzhledu LED diod usnadňují orientaci
4. Vytvoření chyby typu Stuck_atX
5. Vytvoření chyby typu zkrat pomocí operace AND nebo OR
6. Tlačítko pro vytvoření VHDL souboru s danou chybou
7. Seznam signálů, které je možné zkratovat
8. Vytvořená chyba

Při navrhování této aplikace byl kladen důraz na přehledné uspořádání ovládacích prvků a intuitivní postup vytváření poruch. Nechal jsem se inspirovat Průvodci v operačním systému Windows, takže porucha vzniká po jednoduchých krocích. Není možné nic přeskočit, zapomenout nebo naopak udělat jeden krok dvakrát. Ale oproti zmiňovaným Průvodcům je zde jedna výrazná změna – všechno probíhá v jednom okně. Krokování je realizováno postupným zpřístupňováním ovládacích prvků, a přestože se postup větví, není možné provést nějakou nelogickou operaci, např. vytvořit poruchu ShortCut a nezvolit při tom, se kterým signálem se zkratuje. Následující diagram na ukazuje sled kroků při vytváření poruchy:



Obrázek 8: Postup vytváření poruchy

Orientaci usnadňují šipky, které upozorňují uživatele, kde má pokračovat. Jejich vzhled vytváří iluzi LED diod.

Po dokončení vytváření poruchy – její popis se nachází v textovém okénku, viz. Obrázek7, pozice 8 – se zpřístupní tlačítko Vytvořit VHDL (Obrázek 7, pozice 6). Po zmačknutí se otevře klasické dialogové okno pro ukládání souborů. Jméno VHDL souboru je

předpřipraveno – odvozeno od druhu poruchy – ale je možné ho samozřejmě změnit. Pokud uživatel potvrdí uložení souboru, aplikace vytvoří VHDL soubor daného jména, který je dále např. pomocí návrhového systému ISP DesignEXPERT možno zkompilovat a vytvořený model nahrát do FPGA. Algoritmus vytváření VHDL popisu je popsán v následující kapitole.

4.2.2. Rozbor algoritmů

Algoritmy v tomto programu se dělí na ty, které zajišťují vizuální stránku aplikace (např. „rozsvěcení“ šipek, zpřístupňování ovládacích prvků apod.) a na algoritmy, které zajišťují práci se soubory.

Změny okna aplikace korespondují s postupným vytvářením poruchy neboli s diagramem na Obrázku 8. Pokud uživatel udělá krok zpět (např. při téměř dokončené poruše zvolí jiný signál, což je první krok), určité ovládací prvky se zablokují tak, aby nemohlo dojít k omylu. Při posledním kroku se informace o poruše uloží do datové struktury a vygeneruje se její název, který se vypíše do textového okénka. Uvolní se poslední tlačítko Vytvořit VHDL, viz. Obrázek 7, figura 6. Po stisknutí se spustí algoritmus vytváření VHDL souboru.

K uložení informací o chybě je použita datová struktura **Record** – záznam. Nebylo to v tomto případě bezpodmínečně nutné, hodnoty jsou všechny textové, neboli **String**, a bylo by možné je uložit i do proměnných. Vždy je ale, podle mého mínění, výhodné myslet při programování na případné rozšiřování aplikace o další možnosti a v tom případě by to datová struktura typu záznam usnadnila. Například proto, že jde jednoduše uložit do souboru nebo je možné vytvořit dynamické pole poruch (pokud by byl požadavek na vložení více poruch do obvodu). Datová struktura vypadá takto:

```
type
  RChyba = record
        Druh: String;
        Signal: String;
        Operace: String;
        Zkrat: String;
    end;

var
  Chyba: RChyba;
```

K jednotlivým položkám záznamu se přistupuje tečkovou notací, např:

```
 Chyba.Signal := ’n1’;
```

Případně se datová strukturá „otevírá“ příkazem **with** a přistupuje se k položkám záznamu takto:

```
with Chyba do
  begin
    Druh := 'ShortCut';
    Signal := 'y1';
    Operace := 'OR';
    Zkrat := 'b';
  end;
```

Při vývoji algoritmu, který by dokázal zakomponovat nadefinovanou poruchu do VHDL popisu, bylo nutné vyřešit několik dílčích podúkolů. Základní myšlenka je taková, že se využije VHDL popisu originálního obvodu C17, který se změní, případně doplní tak, aby výsledný soubor popisoval obvod C17 s danou poruchou. Důležité samozřejmě je to, aby takto vytvořený soubor *.vhd byl syntakticky správně a bylo ho možné bez dalších úprav zkompilovat a nahrát do programovatelného hradlového pole.

V první řadě bylo nutné najít všechny možné kombinace poruch a signálů a najít způsob, jak tyto poruchy popsat vzhledem ke stávajícímu popisu obvodu. Hlavní problém vyplýval ze samotného jazyka VHDL. Model můžeme klasicky rozdělit na definiční a deklarační část. V první části jsou definovány vstupy, výstupy, vnitřní signály, pomocné signály, konstanty atd. Signály definované jako výstupní však nelze v žádném případě připojit na vstup (kompilátor hlásí chybu). Toto se projeví, pokud se jeden z výstupů obvodu připojuje na trvalou 0 nebo 1, nebo se zkratuje výstup s jiným signálem. Musel jsem toto omezení obejít pomocnými signály, které mohou být vstupní i výstupní, a pomocnými hradly, pro realizaci některých logických operací. Analýzou tohoto problému vzniklo několik pravidel, podle kterých se algoritmus řídí při výběru, jak nadefinovanou chybu popsat. Pravidla jsou názorně ukázána v následující Tabulce 6:

<table>
<tr><th>Porucha</th><th>Signál</th><th>Zkrat</th><th>Pravidlo</th></tr>
<tr><td rowspan="10">Stuck_at 0
Stuck_at 1</td><td>a</td><td>---</td><td rowspan="10">1. (Porucha je Stuck_at 0 OR Stuck_at 1) AND
(signál není y1 OR y2)</td></tr>
<tr><td>b</td><td>---</td></tr>
<tr><td>c</td><td>---</td></tr>
<tr><td>d</td><td>---</td></tr>
<tr><td>e</td><td>---</td></tr>
<tr><td>n1</td><td>---</td></tr>
<tr><td>n2</td><td>---</td></tr>
<tr><td>n3</td><td>---</td></tr>
<tr><td>n4</td><td>---</td></tr>
<tr><td rowspan="2">Stuck_at 0
Stuck_at 1</td><td>y1</td><td>---</td><td rowspan="2">2. (Porucha je Stuck_at 0 OR Stuck_at 1) AND
(signál je y1 OR y2)</td></tr>
<tr><td>y2</td><td>---</td></tr>
<tr><td rowspan="10">ShortCut</td><td>a</td><td rowspan="10">M – {y1, y2}</td><td rowspan="10">3. (Porucha je ShortCut) AND
(signál není y1 OR y2) AND
(zkrat není y1 OR y2)</td></tr>
<tr><td>b</td></tr>
<tr><td>c</td></tr>
<tr><td>d</td></tr>
<tr><td>e</td></tr>
<tr><td>n1</td></tr>
<tr><td>n2</td></tr>
<tr><td>n3</td></tr>
<tr><td>n4</td></tr>
<tr><td>ShortCut</td><td>y1
y2</td><td>M – {y1, y2}</td><td>4. (Porucha je ShortCut) AND
(signál je y1 OR y2) AND
(zkrat není y1 OR y2)</td></tr>
<tr><td rowspan="2">ShortCut</td><td>y1</td><td>y2</td><td rowspan="2">5. (Porucha je ShortCut) AND
(signál je y1 OR y2) AND
(zkrat je y1 OR y2)</td></tr>
<tr><td>y2</td><td>y1</td></tr>
</table>

Tabulka 6: Pravidla pro popis poruch

Poznámky k tabulce:

1) Definujme množinu M = {a, b, c, d, e, n1, n2, n3, n4, y1, y2} jako množinu všech signálů v integrovaném obvodu C17
2) Pravidla 1. a 2. platí pro poruchu Stuck_at 0 stejně jako Stuck_at 1, protože se jedná pouze o nahrazení signálu konstantní hodnotou logická 0 nebo logická 1

3) Pravidlo 4. platí i v případě, že (signál není **y1** OR **y2**) AND (zkrat je **y1** OR **y2**), neboli není důležité, jestliže se zkratuje signál **a** se signálem **y1** nebo signál **y1** se signálem **a**, což vychází z teorie v kapitole 2.

Z tabulky jsou vidět dva zásadní výsledky, a to:

1) Poruchy Stuck_at 0 Stuck_at 1 je možné popsat stejně, mění se pouze název konstanty
2) Pro všechny poruchy je nutné rozlišit, zda se týkají výstupních signálů y1, y2 nebo signálů ostatních. Tento poznatek je ještě rozšířen v případě zkratu.

Příklady pro jednotlivá pravidla:

Pravidlo 1.: Pokud se k signálu z množiny M – {y1, y2} připojuje porucha Stuck_at 0 nebo Stuck_at 1, nahradí se tento signál na vstupech hradel, kde se zvolený signál nachází, konstantou **Stuck_at0** resp. **Stuck_at1**.

Příklad: **n2 / Stuck_at 1**

```
...
BEGIN
  u0: hr_nand PORT MAP (in1=>a, in2=>c, out1=>n1);
  u1: hr_nand PORT MAP (in1=>c, in2=>d, out1=>stoka);
  u2: hr_nand PORT MAP (in1=>b, in2=>Stuck_at0, out1=>n3);
  u3: hr_nand PORT MAP (in1=>Stuck_at0, in2=>e, out1=>n4);
  u4: hr_nand PORT MAP (in1=>n1, in2=>n3, out1=>y1);
  u5: hr_nand PORT MAP (in1=>n3, in2=>n4, out1=>y2);
END struct;
```

Na výstupu hradla `u1` byl původně signál `n2`. Jelikož se ale signál `n2` dále v obvodu nahrazuje konstantní logickou 0, je výstup tohoto hradla poslán do virtuální „stoky“.

Pravidlo 2.: Pokud se definuje porucha Stuck_at 0 nebo Stuck_at 1 na signálech `y1` nebo `y2`, v chování obvodu to znamená, že výstup zasažený touto poruchou nelze nijak měnit, je zaseklý na dané logické hodnotě. Jelikož jsou ale signály `y1` a `y2` definovány jako výstupní, nelze jim přímo přiřadit konstantní hodnotu. Využil jsem k tomu následující trik:

Příklad: **y1 / Stuck_at 0**

```
...
BEGIN
  u0: hr_nand PORT MAP (in1=>a, in2=>c, out1=>n1);
  u1: hr_nand PORT MAP (in1=>c, in2=>d, out1=>n2);
  u2: hr_nand PORT MAP (in1=>b, in2=>n2, out1=>n3);
  u3: hr_nand PORT MAP (in1=>n2, in2=>e, out1=>n4);
```

```
  u4: hr_nand PORT MAP (in1=>n1, in2=>n3, out1=>stoka);
  u5: hr_nand PORT MAP (in1=>n3, in2=>n4, out1=>y2);
  ch <= Stuck_at0;
  u6: hr_and PORT MAP (in1=>ch, in2=>ch, out1=>y1);
END struct;
```

Výstup hradla u4, který byl původně y1 je poslán do „stoky“. Do pomocné proměnné ch se přiřadí příslušná konstanta. Na výstupní signál y1 se dostane logickou operací AND realizovanou hradlem typu AND.

Pravidlo 3.: Pokud jsou mezi sebou zkratovány signály z množiny M – {y1, y2}, je situace poměrně jednoduchá. Signály, které se zkratují, se přivedou na vstupy hradla realizujícího logickou operaci, se kterou je zkrat spojen (AND nebo OR). Výstup tohoto hradla je přiřazen do pomocné proměnné ch. Tou jsou následně nahrazeny vstupy hradel obvodu, kde se nachází zkratované signály.

Příklad: **d / ShortCut OR e**

```
...
BEGIN
-- zkrat
  u6: hr_or  PORT MAP (in1=>d, in2=>e, out1=>ch);

-- C17 se zkratem
  u0: hr_nand PORT MAP (in1=>a, in2=>c, out1=>n1);
  u1: hr_nand PORT MAP (in1=>c, in2=>ch, out1=>n2);
  u2: hr_nand PORT MAP (in1=>b, in2=>n2, out1=>n3);
  u3: hr_nand PORT MAP (in1=>n2, in2=>ch, out1=>n4);
  u4: hr_nand PORT MAP (in1=>n1, in2=>n3, out1=>y1);
     u5: hr_nand PORT MAP (in1=>n3, in2=>n4, out1=>y2);
END struct;
```

Pravidlo 4.: Poněkud složitější situace nastává, pokud se zkrat týká jednoho z výstupních vodičů y1 nebo y2. Zde se nedá využít možnosti, přivést zkratované signály na vstupy hradla realizujícího logickou operaci zkratu. K řešení tohoto problému jsem použil podobného postupu, jako v Pravidle 2.

Příklad: **b / ShortCut AND y1**

```
...
BEGIN
-- zkrat signálů b - y1
  u6: hr_and  PORT MAP (in1=>b, in2=>xy1, out1=>ch);
  u7: hr_and  PORT MAP (in1=>ch, in2=>ch, out1=>y1);
```

```
-- C17 se zkratem
  u0: hr_nand PORT MAP (in1=>a, in2=>c, out1=>n1);
  u1: hr_nand PORT MAP (in1=>c, in2=>d, out1=>n2);
  u2: hr_nand PORT MAP (in1=>ch, in2=>n2, out1=>n3);
  u3: hr_nand PORT MAP (in1=>n2, in2=>e, out1=>n4);
  u4: hr_nand PORT MAP (in1=>n1, in2=>n3, out1=>xy1);
  u5: hr_nand PORT MAP (in1=>n3, in2=>n4, out1=>y2);
END struct;
```

Nejdříve se výstup z hradla `u4` přejmenuje na `xy1`. Tento pomocný signál není pevně definován jako vstupní nebo výstupní, takže ho lze přivést, společně s druhým zkratovaným signálem na vstup hradla AND, které provede požadovanou logickou operaci zkratu. Vznikne výsledná chyba v pomocné proměnné `ch`. Tou se pak nahradí signál `b` na všech vstupech hradel obvodu, v tomto případě jen `u2`. Ještě je nutné dostat výsledek zkratu na výstup `y1`. To je provedeno pomocí triku z Pravidla 2.

Pravidlo 5.: Poslední možností, která může nastat, je zkrat obou výstupních signálů. K popisu tohoto případu je použit kombinace postupů z Pravidla 2 a Pravidla 4.

Příklad: **y1 / ShortCut OR y2**

```
...
BEGIN
-- zkrat signálů y1 - y2
  u6: hr_or  PORT MAP (in1=>xy1, in2=>xy2, out1=>ch);
  u7: hr_and  PORT MAP (in1=>ch, in2=>ch, out1=>y1);
  u8: hr_and  PORT MAP (in1=>ch, in2=>ch, out1=>y2);

-- C17 se zkratem
  u0: hr_nand PORT MAP (in1=>a, in2=>c, out1=>n1);
  u1: hr_nand PORT MAP (in1=>c, in2=>d, out1=>n2);
  u2: hr_nand PORT MAP (in1=>b, in2=>n2, out1=>n3);
  u3: hr_nand PORT MAP (in1=>n2, in2=>e, out1=>n4);
  u4: hr_nand PORT MAP (in1=>n1, in2=>n3, out1=>xy1);
  u5: hr_nand PORT MAP (in1=>n3, in2=>n4, out1=>xy2);
END struct;
```

Oba výstupy hradel `u4` a `u5` se přejmenují na pomocné proměnné `xy1` a `xy2`. Dále se zkratují hradlem příslušné logické operace, z čehož vznikne výsledek `ch`. Nakonec se logickou operací AND přivede chyba `ch` na výstupní signály `y1` a `y2`.

Poté, co jsem stanovil těchto 5 pravidel a vymyslel způsob, jak všechny kombinace chyb a signálů popsat, bylo nutné algoritmus přepsat do kódu. V dalším textu je proveden rozbor funkcí a procedur souvisejících s vytvářením VHDL souborů.

Když se snažím naprogramovat řešení nějakého problému, hledám nejdříve způsob, jak výsledku dosáhnout „ručně“. V případě mojí diplomové práce šlo o to, pročíst originální VHDL popis obvodu C17 – obecně řečeno textový soubor – a doplnit do něj několik řádků a přepsat některá písmena. Následně považuji za vhodné pojmenovat úkony, kterými jsem se k výsledku dostal:

- vložit libovolný počet řádků textu na určité místo v textu
- přepsat určité slovo

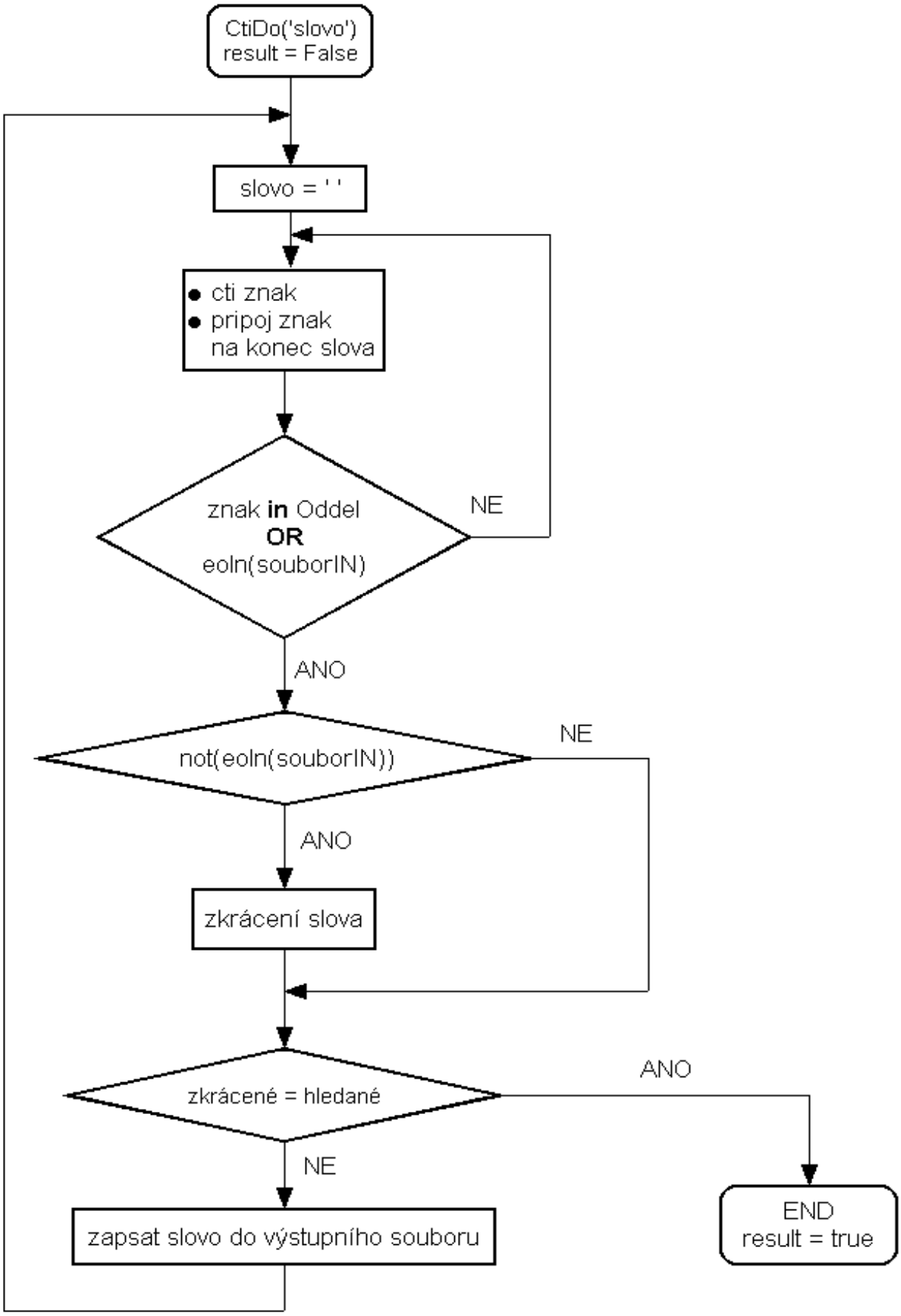
K oběma těmto úkonům bylo nejdříve potřebova nalézt místo, kde se mají provést. K tomu účelu vznikla funkce, která dokáže prohledávat libovolný textový soubor a najít v něm hledané slovo – posloupnost znaků. Důležité je v tomto případě poznat, kde slovo začíná a končí. K tomu účelu byla nadefinována množina oddělovačů:

```
var
  Oddel: Set of Char;
begin
  Oddel := [' ', ';', ',', ':', '(', ')'];
...
```

Algoritmus funkce CtiDo(slovo:String): Boolean; vysvětluje následující vývojový diagram. Vstupním parametrem funkce je hledaný řetězec. Funkce pracuje s dvěma soubory, VHDLin jako čtecí a VHDLout jako zapisovací. Na začátku je výsledná hodnota funkce nastavena na False. Čte se vždy řádek od začátku do konce a přečtené znaky se skládají do slova, dokud se nenarazí na oddělovač nebo konec řádku. Pokud takto rozpoznané slovo končí oddělovačem, tak se tento na konci umaže. Hledá se totiž např. slovo **END** a ne slovo **END;**. Provede se porovnání, zda nalezené slovo odpovídá hledanému. Pokud ano, funkce končí s výsledkem True. Pokud ne, zapíše se nalezené slovo do výstupního souboru.

Pokud funkce skončí s výsledkem True, zastavil se „kurzor“ před hledaným slovem a může dojít k jeho přepsání. Pokud se hledané slovo v souboru nenachází, dojde k prostému okopírování celého souboru VHDLin do VHDLout.

Při práci se soubory, zejména u těch, u kterých předem není známa jejich přesná struktura, je vhodné využít **Zpracování výjimek**, viz. kapitola 3.1.1. V tomto konkrétním případě může dojít k tomu, že soubory nejsou otevřeny, čtecí soubor byl na disku fyzicky vymazán během hledání slova, apod.



Obrázek 9: Vývojový diagram funkce CtiDo(slovo:String):Boolean;

Nyní jsem napsal proceduru, která prakticky provádí vytváření výstupního VHDL souboru. Vstupem je datová struktura obsahující informace o chybě. Nejdříve je nutné rozhodnout, ke kterému pravidlu porucha patří. To se provádí postupným vyhodnocováním podmínek z tabulky 6. Tím je jasně rozhodnuto, jak se má změnit originální VHDL popis. Nejjednodušší případ nastává, pokud se jedná o poruchu patřící k Pravidlu 1. Průběh vytváření VHDL souboru popisuje následující vývojový diagram na Obrázku 10. Jedná se o chybu **n2 / Stuck_at1**, která je popsána v příkladě pro Pravidlo 1. Datová struktura **Chyba** potom vypadá takto:

```
Chyba.Signal := ‘n2’;
Chyba.Druh := ‘Stuck_at1’;
Chyba.Operace := ‘’;
Chyba.Zkrat := ‘’;
```

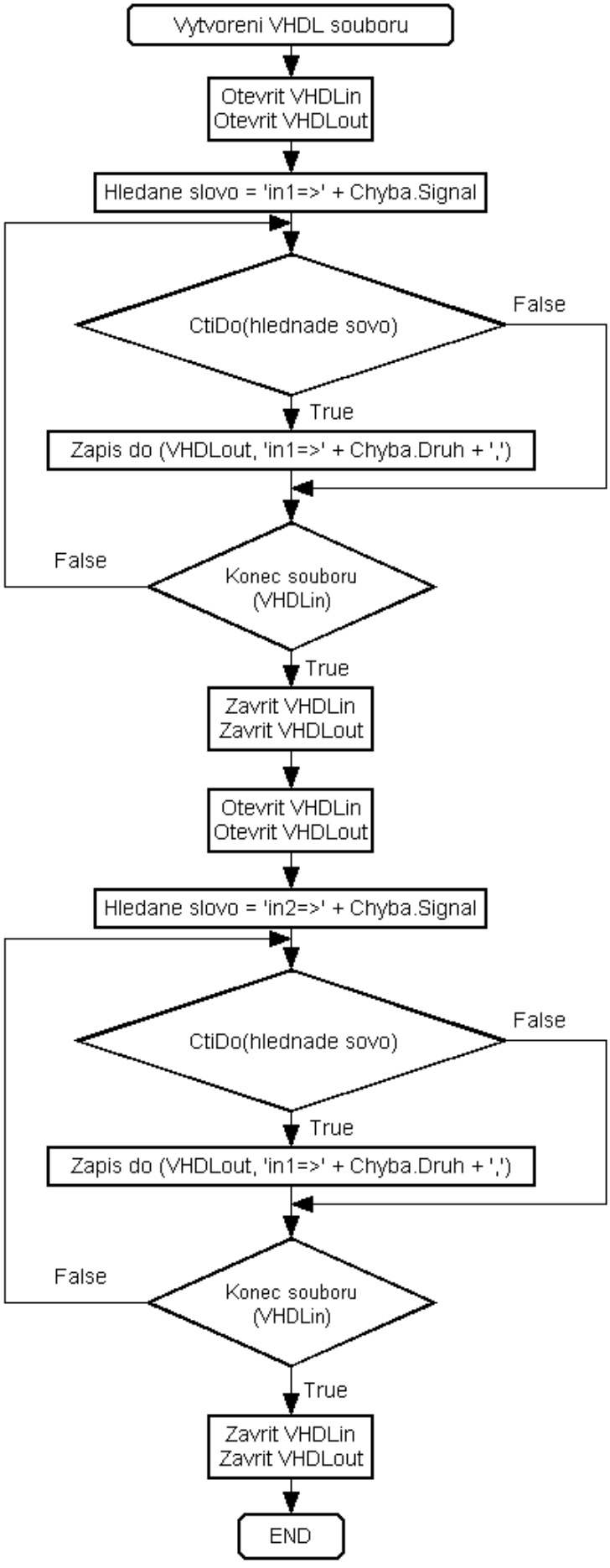
Všechny čtyři položky záznamu **Chyba** jsou typu String, neboli textový řetězec. Navrhl jsem je takto proto, že se tyto hodnoty mohou přímo zapisovat do výstupního textového souboru, je z nich možné vytvořit název poruchy a mimo jiné také název souboru.

Popis obvodu s poruchou je odvozen od originálního popisu bez poruchy, soubor C17_OK.vhd. V uvedeného příkladu je vidět, že kolik **změn** je potřeba oproti originálnímu souboru provést, tolik průchodů souborem je nutné udělat. Přesněji řečeno, za jednu **změnu** se považuje prohledání celého souboru a nahrazení jednoho hledaného slova na všech místech, kde se nachází, jiným. To znamená, že v příkladě, který jsem zde uvedl (chyba n2 / Stuck_at1) se nejdříve hledá slovo `` `in1=>n2‘ `` a je nahrazeno slovem `` `in1=>Stuck_at1‘ ``, tj. první průchod, a následně slovo `` `in2=>n2` ``, které je přepsáno slovem `` `in2=>Stuck_at1‘ ``, což jsou celkem dva průchody. Jedním průchodem je také možno přidat libovolný počet řádků na jakékoli místo. Následující Tabulka 7 ukazuje, kolik průchodů je třeba provést pro různé druhy poruch a signálů.

Pravidlo	Poruchy	Signály	Zkrat	Počet průchodů
1	Stuck_at0, Stuck_at1	M – {y1, y2}	---	2
2	Stuck_at0, Stuck_at1	y1, y2	---	1
3	ShortCut	M – {y1, y2}	M – {y1, y2}	5
4	ShortCut	M – {y1, y2}	y1, y2	4
5	ShortCut	y1	y2	2

Tabulka 7: Shrnutí pravidel pro vkládání poruch

Algoritmus by bylo možné samozřejmě napsat i tak, aby se celá změna provedla během jednoho průchodu souborem. Ovšem celý postup by se tím značně zkomplikoval a znepřehlednil. Soubory, se kterými zde pracuji jsou velikosti přibližně 1kB. Rozdíl v rychlosti algoritmu, pokud se otevře, přečte a zavře takovýto malý soubor jednou nebo desetkrát při rychlosti dnešních počítačů prakticky neexistuje.



Obrázek 10: Vývojový diagram procedury vytváření VHDL souboru

4.3. Aplikace C17

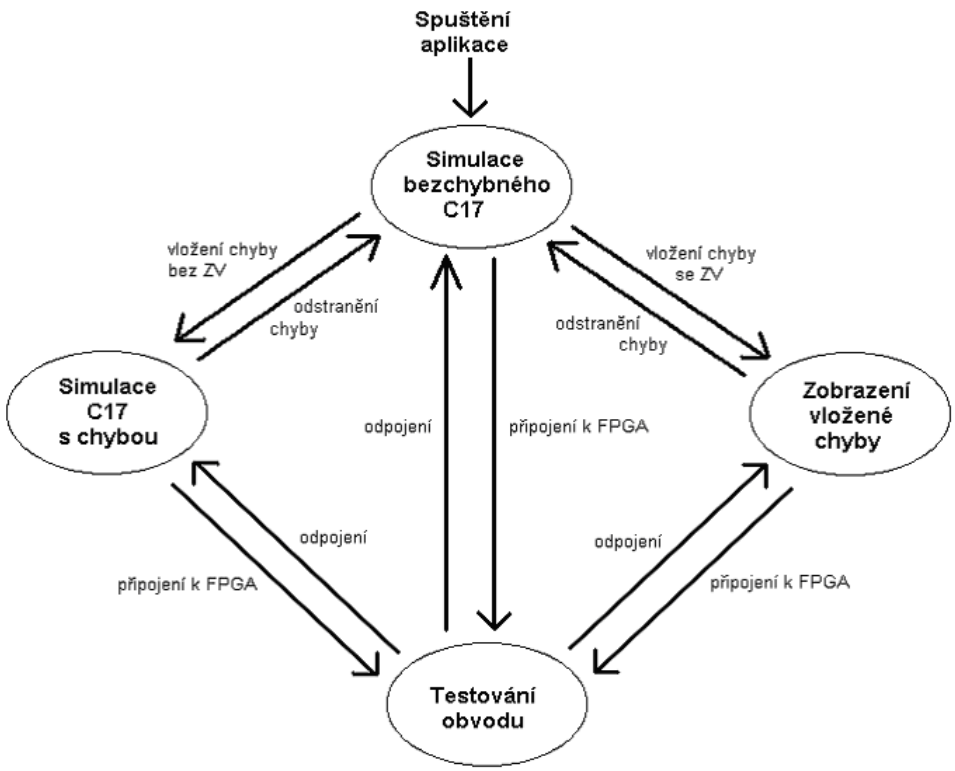
4.3.1. Manuál

4.3.1.1. Úvod

Aplikace C17 – SIMULATOR se dá považovat za výukovou aplikaci detailně se zabývající výhradně číslicovým obvodem C17. Dokáže tento obvod simulovat v bezporuchovém stavu, simulovat většinu poruch, které mohou v obvodu nastat, rozlišovat zkraty se zpětnou vazbou, komunikovat paralelním portem s modelem obvodu nahraném v reálném hradlovém poli a porovnávat rozdíly mezi simulovaným a reálným obvodem a v neposlední řadě vyhodnocovat a graficky zobrazovat výsledky. Dále je v aplikaci implementováno několik možností nastavení vzhledu a chování programu a také poměrně rozsáhlé nastavování propojení aplikace potažmo počítače s FPGA Lattice po paralelním kabelu.

4.3.1.2. Základní používání

Program se může nacházet ve čtyřech různých stavech, mezi kterými přechází podle činností, které s ním uživatel vykonává. Názorně to ukazuje následující stavový diagram:



Obrázek 11: Stavový diagram aplikace C17 Simulator

Simulace bezchybného C17

Do tohoto stavu se aplikace dostane vždy po spuštění, lze ho tedy nazvat též stavem základním. Kromě nastavovacích funkcí, které jsou dostupné během celého používání aplikace, je zde dále možné:

- simulovat obvod C17 v bezporuchové stavu
- pomocí **Nástroje pro vkládání simulovaných poruch** vložit do virtuálního obvodu poruchu
- připojit se přes LPT k obvodu C17 nahraném v reálném hradlovém poli

Poslední dvě operace způsobí přechod do jednoho z následujících stavů.

Simulace C17 s poruchou

Pokud vložíme do obvodu poruchu typu Stuck_at 0, Stuck_at 1 nebo ShortCut, který neobsahuje zpětnou vazbu, viz. Kapitola 3, aplikace simuluje, jak se takový obvod bude chovat. Signál, kterého se porucha týká je barevně označen a pokud na jeho popisek uživatel najede kurzorem myši, zobrazí se popis příslušné chyby. Dále je možno provést:

- vloženou poruchu odstranit
- změnit poruchu na jinou
- propojit aplikaci s hradlovým polem ispLSI2032

Zobrazení vložené chyby

Pokud chybou, kterou uživatel pro simulaci vybral, je zkrat obsahující zpětnou vazbu, viz. Kapitola 3, není možné aplikací C17 Simulator chování obvodu simulovat. Aplikace přestane určovat a zobrazovat vnitřní stavy a předpokládané výstupy, ale signály, které jsou zkratovány se barevně ve schématu vyznačí. Další možný postup je:

- vloženou poruchu odstranit
- změnit chybu na jinou
- propojit aplikaci s FPGA

Testování obvodu

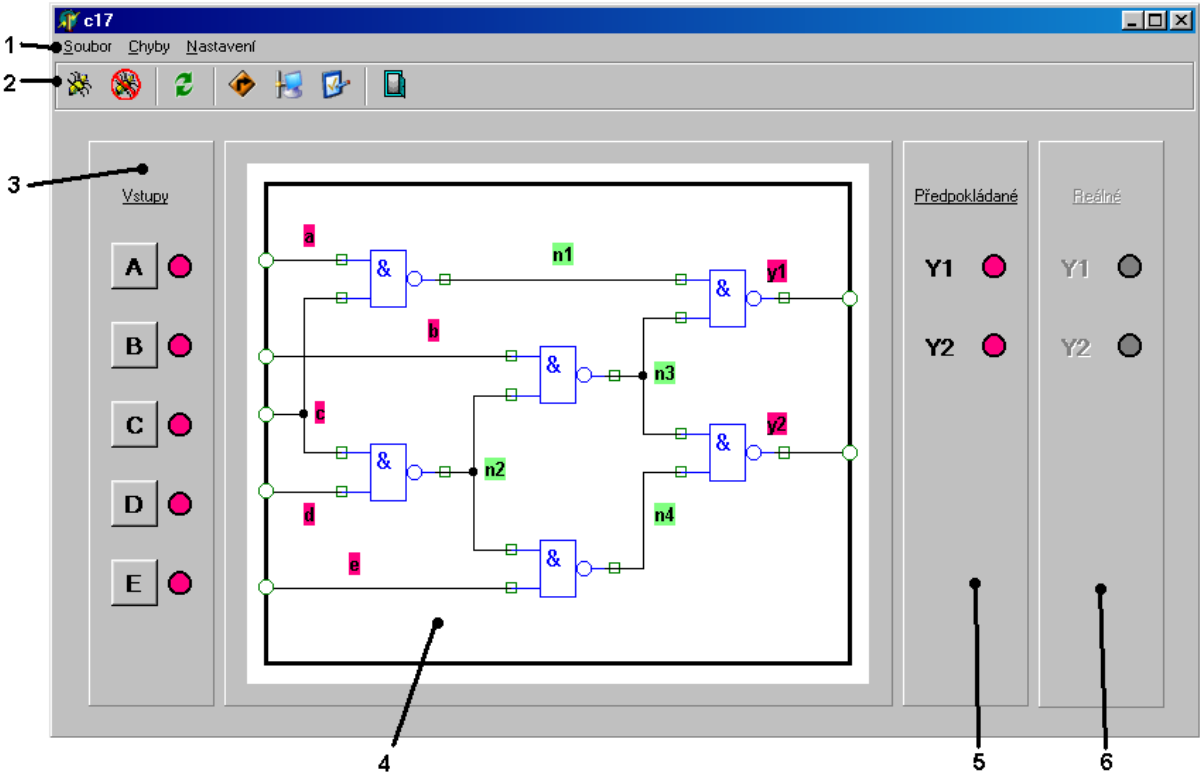
Před tím, než může aplikace komunikovat s hradlovým polem, musí být správně nastavena paralelní komunikace. Postup je popsán v následující kapitole 4.4.1.3.

Po nastavení LPT může uživatel provádět testování reálného obvodu C17, resp. porovnávání simulovaného a reálného obvodu. V případě, že výchozím stavem testování obvodu byl stav **Simulace bezchybného C17** nebo **Simulace C17 s chybou**, pak program

simuluje obvod a zároveň posílá hodnoty vkládaných vstupů do reálného obvodu. Následně čte jeho výstupy, které zobrazuje jako *reálné výstupy*. V případě zkratu se zpětnou vazbou, neboli výchozího stavu **Zobrazení vložené chyby** pouze posílá data ze vstupů a čte výstupy reálného obvodu, výsledky zobrazuje, ale není možné je porovnávat s výsledky simulace.

Po ukončení spojení, případně vinou jeho přerušení (např. odpojení kabelu) přejde aplikace zpět do stavu, ve kterém se nacházela před připojením.

Ovládání aplikace jsem se snažil přiblížit zaběhnutým standardům v profesionálních aplikacích. Nejčastěji používané operace jsou přístupné pomocí Panelu nástrojů, v klasickém Menu jsou k dispozici všechny možnosti, které C17 Simulator nabízí.



Obrázek 12: Hlavní okno aplikace

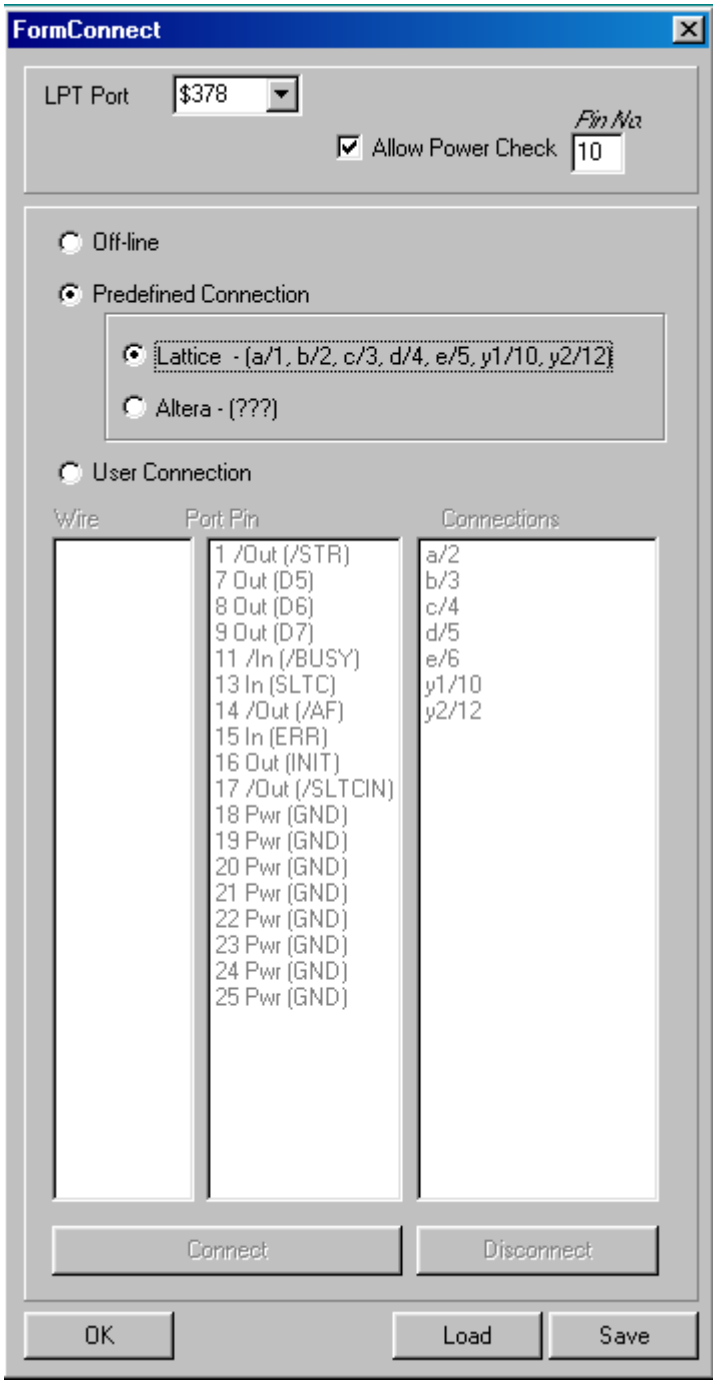
1. Menu
2. Panel nástrojů
3. Panel s tlačítky pro vkládání vstupů
4. Aktivní schéma, zobrazující logické úrovně jednotlivých signálů
5. Panel s předpokládanými výstupy (odvozené simulací)
6. Panel s reálnými výstupy (přečtené z reálného obvodu)

Ad 2. Panel nástrojů:

1) Simulovat poruchu
2) Odstranit simulovanou poruchu
3) Refresh obrazovky
4) Evropska/US norma logických hradel
5) Nastavení připojení LPT
6) Nastavení aplikace
7) Ukončení aplikace

4.3.1.3. Nastavení paralelní komunikace

LPT komunikace mezi C17 Simulatorem a reálným obvodem se nastavuje poměrně detailně. Základem je ovšem to, že by aplikace z pohledu komunikace měla být robustní, a to vzhledem k systému, na kterém je spuštěna, tak i k zásahům uživatele. Program je vyvinut pro operační systém Windows a měl by být stabilní od starých Windows 95, přes různé verze Windows 98 až po moderní Windows XP. Záměrně však používám slovního spojení „měl by být“, protože jsem provedl testování pouze na třech počítačích, které obsahovali operační systém Windows 98 SE a Windows XP Professional Corporation Edition.



obrázek 13: Okno nastavení komunikace přes paralelní port

Prvním krokem při nastavování paralelního portu je jeho výběr. Jak je detailněji uvedeno v kapitole 3.5., v běžném PC je možné najít tři LPT porty, z čehož nejčastěji užívaným je LPT1 – klasický LPT port pro tiskárnu. Jeho bázová adresa je $378. Druhý nejpoužívanější port LPT2 má adresu $278.

Dále je nutné jednotlivé vstupní a výstupní signály přiřadit pinům paralelního portu. Po spuštění aplikace, a tedy i při prvním nastavování je komunikace ve stavu Off-line. Pro nejpoužívanější FPGA obvody od firem Lattice a Altera jsou připravená *Předpřipravená propojení*. Po kliknutí na jednu z těchto možností se propojení automaticky nastaví. Je ovšem

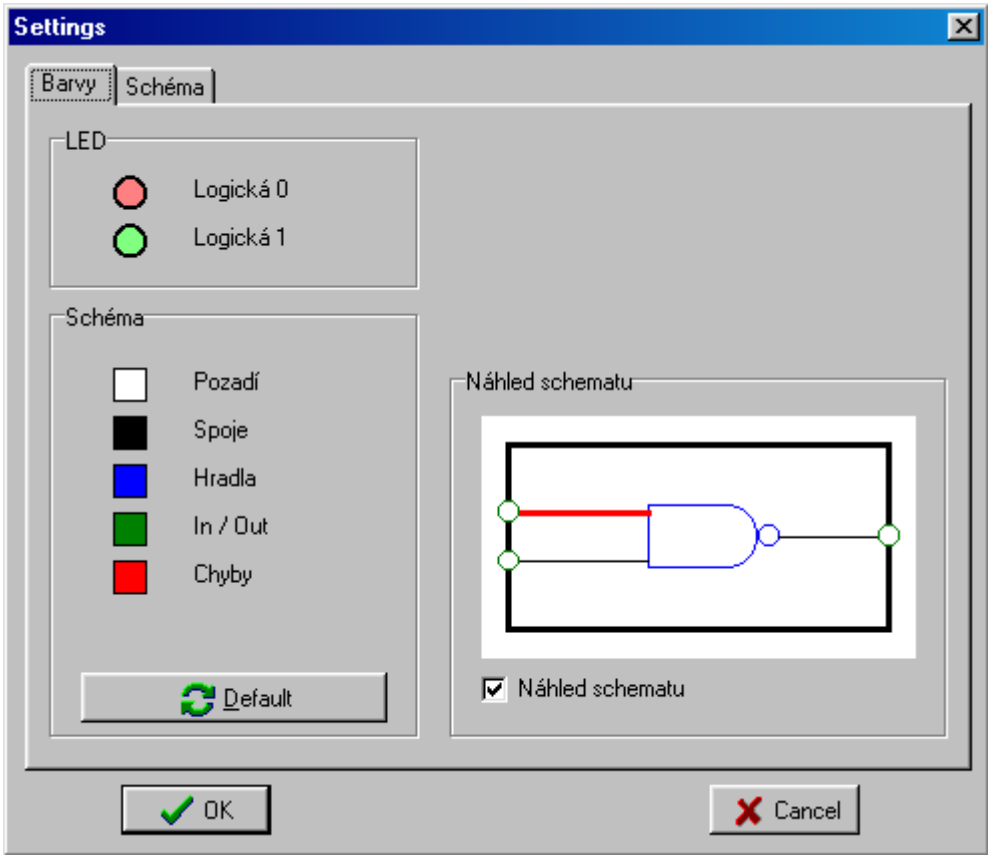
také možné nastavit si propojení vlastní, pokud je zvoleno *Uživatelské propojení*. Potom je na uživateli samotném, jak signály s piny LPT propojí. Pro správné fungování komunikace je ovšem nutné dodržovat následující zásady:

- **výstupní signály** (označené *out*) musí být připojené na výstupní piny
- **vstupní signály** (označené *in*) musí být připojené na vstupní piny
- **piny označené Pwr (GND)** jsou spojeny a uzemněny, a není je proto možné využít ani jako vstupní nebo výstupní
- **invertující piny** jsou označeny lomítkem /. Mezi použití invertujících a neinvertujících pinů není žádný rozdíl. Při posílání dat přes invertující piny se bity automaticky převrací, aby do reálného obvodu vstupovaly tak, jak jsou doopravdy myšleny.

Pokud je jedna z těchto zásad při propojování porušena, je příslušné propojení označeno (Err !!) a ozve se varovný zvuk. Když nedojde k opravení této chyby nebo uživatel nezapojí všechny signály na piny, zůstává aplikace v režimu Off-line, případně do něj přejde, pokud předtím bylo propojení funkční.

4.3.1.4. Nastavení vzhledu aplikace

Jelikož významná část aplikace zajišťuje vizualizaci chování obvodu C17, implementoval jsem několik možností, jak její vzhled upravovat. Přestože tyto funkce nejsou nutné pro základní funkci programu, simulaci a komunikaci, jsou běžné v profesionálních aplikacích a řekl bych, že jsou považovány za standard.



obrázek 14: Okno nastavení vzhledu aplikace

Barva virtuálních LED pro hodnoty logická 0 a logická 1 se nastavují společně. Zvolené barvy jsou použity i pro pozadí popisů signálů ve schématu a pro okno *Vnitřní stavy*. Dále je možné měnit barevné vlastnosti schématu. Před tím, než uživatel změny potvrdí si je může prohlédnout na náhledu.

Druhá karta nastavovacího okna se týká možností zobrazení. Je možné vypnout nebo zapnout okno s logickými hodnotami vnitřních stavů obvodu a zviditelnit nebo naopak skrýt barevné popisky u všech signálů. Těchto možností by teoreticky bylo možné využít pro procvičování studentů, pokud by moje aplikace byla nasazena do výuky.

Nakonec jsem ještě na žádost vedoucího diplomové práce doplnil možnost přepnutí mezi zobrazováním hradel podle evropské a americké normy.

4.3.1.5. Simulace poruchy

Vytváření simulované poruchy V C17 Simulator probíhá v podobném duchu, jako v aplikaci C17 Bugger. Po dokončení vytvoření poruchy se zasažené signály ve schématu barevně označí (barvu lze nastavit v Nastavení aplikace, viz. kapitola 4.4.1.4.). V kapitole 2, Teorie poruch v IO, je podrobně vyložen rozdíl mezi zkraty se zpětnou vazbou a bez zpětné

vazby. Pro zjednodušení C17 Simulator simuluje jen ty obvody, které je možné popsat pravdivostní tabulkou. Do této množiny patří kromě obvodu bez poruchy ještě obvody s poruchami typu Stuck_at 0, Stuck_at 1 a ShortCut bez zpětné vazby.

Simulací obvodů s poruchou se dá využít při práci s reálným obvodem, který také obsahuje poruchu. V některých případech je například možné zjistit, která porucha je v neznámém reálném obvodu. To lze, pokud chyba, projev poruchy v chování obvodu, je unikátní. Tento případ ovšem u obvodu, který má pět vstupů, ale pouze dva výstupy, není příliš častý. Většinou se poruchy maskují, což znamená, že pro stejné vstupy se dvě a více poruch projeví na výstupu stejně a není je z toho důvodu možné rozlišit.

4.3.2. Rozbor algorimtů

Algoritmy v aplikaci C17 Simulator se dělí do čtyř skupin:

1) obsluha událostí komponent
2) simulace
3) vizualizace
4) komunikace

4.3.2.1. Obsluha událostí komponent

Tyto algoritmy patří k nejpočetnějším a zároveň nejjednodušším algoritmům, které jsou pevně svázány se samotnou podstatou vývojového prostředí Delphi a s programování pod operačním systémem Windows vůbec. Ošetřují události vyvolané interakcí uživatele s ovládacími prvky aplikace, např. stisknutí tlačítka, pohyb myši nad určitým objektem, zvolení položky menu apod. Realizují buďto jednoduchou operaci (např. nastavení logické proměnné na False/True) nebo spouštějí *akce* obsahující složitější algoritmy. *Akce – Actions –* jsou ve své podstatě procedury a funkce, nebo jejich sledy, stejně jako v TurboPascalu pro DOS. V Delphi se doporučuje, spíše než obyčejných procedur, využívat právě zmíněných *akcí* a to jednak proto, že Delphi zajišťují, aby byly pro Windows „stravitelnější“ a zároveň podporují možnosti (např. seskupování akcí a spouštění po skupinách), které by jinak musel vývojář programovat.

4.3.2.2. Simulace

Algoritmy simulace obvodu provádějí v podstatě logické operace jednotlivých hradel, od vstupů postupně až k výstupům. Pokud by cílem bylo emulovat pouze chování obvodu

C17 bez poruchy, šlo by k tomu poměrně jednoduše využít pravdivostní tabulky. V programu je ovšem implementována simulace velké většiny poruch, kterými se v diplomové práci zabývám. Oproti původnímu zadání je to poměrně rozsáhlé rozšíření. Když se spočítají všechny kombinace druhů poruch a signálů dostane se následující výsledek (uvažovány jsou poruchy Stuck_at 0, Stuck_at 1, ShortCut bez zpětné vazby a 11 signálů a, b, c, d, e, n1, n2, n3, n4, y1 a y2):

Porucha	**Počet kombinací chyby a signálů**
Stuck_at 0	11
Stuck_at 1	11
ShortCut bez ZV	40
Součet	**61**

Tabulka 8: Počet všech kombinací poruch a signálů

Poznámka k Tabulce 8: U poruch typu ShortCut se např. a ShortCut b bere jako identická porucha s b ShortCut a. Je počítána jen jednou.

To by znamenalo vytvoření minimálně 61 pravdivostních tabulek. Tato metoda by byla jistě mnohonásobně rychlejší, než provádění logických operací a vyhodnocování dotazů `if ... then`. Stačilo by vyhledat příslušnou tabulku podle druhu chyby a pomocí ní pak „simulovat“ – dáli se tomu vůbec v takovém případě tak říkat – číslicový obvod. Pravda je ale taková, že o rychlost v této aplikaci vůbec nejde, zvláště když se přihlédne ke kmitočtům dnešní procesorů a jednoduchosti požadovaných operací, je rychlost více než dostačující. Podstatnější důvod je ten, že vytvoření takového počtu pravdivostních tabulek by bylo pracné, časově náročné a odhalení případných chyb v nich takřka nemožné.

Analyzoval jsem problém, zkraty kterých signálů zapojují zpětnou vazbu, abych je mohl ze simulace vyloučit. Výsledek jsem sestavil do tabulky, která je uvedena v Příloze C.

Číslicový obvod C17 je tedy realizován sekvencí následujících logických operací:

```
n1 := NOT(a AND c);
n2 := NOT(c AND d);
n3 := NOT(b AND n2);
n4 := NOT(n2 AND e);
y1 := NOT(n1 AND n3);
y2 := NOT(n3 AND n4);
```

Z důvodu možnosti přítomnosti chyby na jednom nebo dvou (v případě zkratu) signálech je ovšem nutné toto prověřovat pro vstupní signály logické operace před tím, než se provede. Pro výstupní signály `y1` a `y2` je ještě nutné kontrolovat přítomnost chyby na konci algoritmu. Tím vznikne jen trochu složitější algoritmus:

```
if (Chyba.Signal = 'a') OR (Chyba.Zkrat = 'a') then
  a := ProvedChybu(a);
if (Chyba.Signal = 'c') OR (Chyba.Zkrat = 'c') then
  c := ProvedChybu(c);
n1 := NOT(a AND c);

if (Chyba.Signal = 'd') OR (Chyba.Zkrat = 'd') then
  d := ProvedChybu(d);
n2 := NOT(c AND d);

if (Chyba.Signal = 'b') OR (Chyba.Zkrat = 'b') then
  b := ProvedChybu(b);
if (Chyba.Signal = 'n2') OR (Chyba.Zkrat = 'n2') then
  n2 := ProvedChybu(n2);
n3 := NOT(b AND n2);


if (Chyba.Signal = 'e') OR (Chyba.Zkrat = 'e') then
  e := ProvedChybu(e);
n4 := NOT(n2 AND e);

if (Chyba.Signal = 'n1') OR (Chyba.Zkrat = 'n1') then
  n1 := ProvedChybu(n1);
if (Chyba.Signal = 'n3') OR (Chyba.Zkrat = 'n3') then
  n3 := ProvedChybu(n3);
y1 := NOT(n1 AND n3);

if (Chyba.Signal = 'n4') OR (Chyba.Zkrat = 'n4') then
  n4 := ProvedChybu(n4);
y2 := NOT(n3 AND n4);

if (Chyba.Signal = 'y1') OR (Chyba.Zkrat = '') then
  y1 := ProvedChybu(y1);
if (Chyba.Signal = 'y2') OR (Chyba.Zkrat = '') then
  y2 := ProvedChybu(y2);
```

Z kódu je vidět, že pokud je signál zasažen poruchou, volá se funkce `ProvedChybu` s parametrem *název signálu*. Ta rozliší, o kterou chybu se jedná, najde její výsledný projev a přiřadí tuto hodnotu zpět do signálu.

4.3.2.3. Vizualizace

Algoritmy vizualizační zajišťují jednak vykreslování schématu obvodu a také zobrazování výsledků simulace a komunikace.

Schéma obvodu je kresleno na *Canvas* – plochu – komponenty *TImage*, která umožňuje metody jako úsečka, kružnice, obdélník, psaní textu apod. Před vykreslením se plocha smaže. Další kroky se provádí v pořadí:

- vykreslení hradel, v normě:
  - evropské
  - americké
- vykreslení spojů
- označení spojů zasažených poruchou
- vykreslení uzlů
- vykreslení „sdířek“ vstupů a výstupů
- obarvení popisků signálů
- zobrazení / skrytí popisků signálů

Každý objekt (viz. výše uvedené body kromě posledních dvou) má referenční bod, od kterého je vykreslován. Seznamy těchto bodů jsou uloženy v datových souborech, které se načítají při spůštění aplikace. K vytvoření těchto souborů jsem napsal jednoduchý program pro vkládání souřadnic. Barva jednotlivých objektů se nastavuje viz. kapitola 4.4.1.4.

Logické hodnoty vstupních signálů, vnitřních stavů a simulovaných i reálných výstupů jsou pojaté jako virtuální dvoubarevné LED. Pokaždé, když dojde ke změně jednoho ze vstupních bitů se nejdříve provedou příslušné kroky simulace a komunikace a poté se tyto „LEDky“ zbarví podle výsledků. Podobným principem se mění i pozadí u popisků signálů ve schématu. Barvy pro obě logické úrovně lze nastavit přes okno Nastavní vzhledu aplikace, kapitola 4.4.1.4.

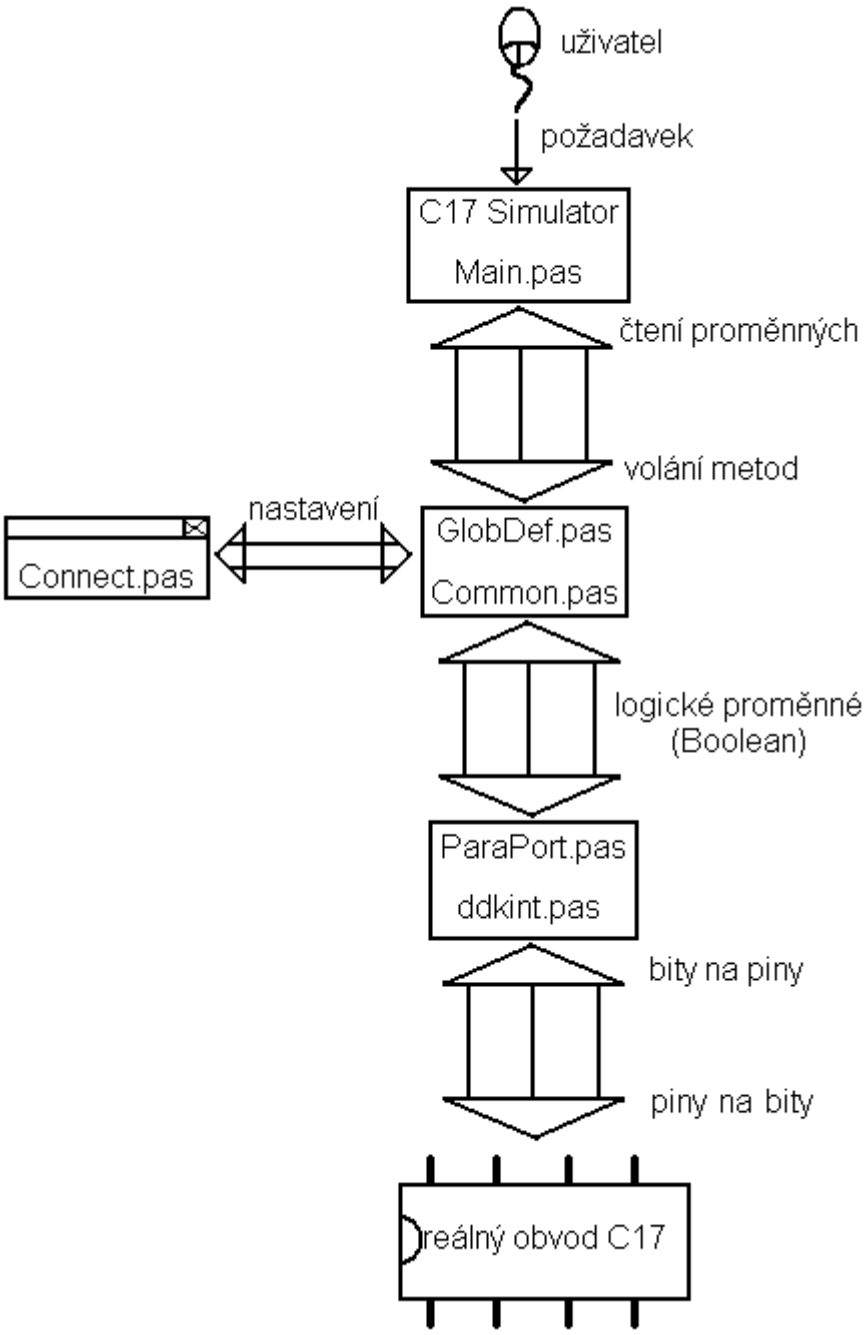
4.3.2.4. Komunikace

Aplikace spouští procesy komunikace ve chvíli, kdy je splněno několik podmínek: PC a reálný obvod (FPGA Lattice, Altera, ...) jsou propojeny paralelním kabelem a LPT je správně nastaveno, což je vysvětleno v kapitole 4.4.1.3. Očekává se, že je připojen obvod C17 (jeho model je nahrán v FPGA) a že uživatel dokáže propojit piny LPT portu se správnými piny obvodu. Chyby v zapojení nebo vůbec použití obvodu C17 by teoreticky bylo možné testovat, pokud by se jednalo pouze o obvod neobsahující poruchy. Není to ovšem prakticky možné u obvodů s poruchami. Na druhou stranu je aplikace určena pro výuku studentů, kteří by měli být seznámeni s problematikou a hlavním účelem je, aby si mohli věc vyzkoušet, aniž by museli detailně proniknout do témat, které s látkou o číslicových obvodech nemají příliš společného.

Pokud je tedy komunikace ve stavu online, posílají se při každé změně vstupních signálů data do reálného obvodu a následně se čtou jeho výstupy. Nejnižší level komunikace pracující s jednotlivými bity na úrovni assembleru a driverů paralelního portu provádí unity *ParaPort.pas* a *DdKint.pas*, které vytvořil Alexandr Zloba z firmy SpecoSoft (http://www.specosoft.com) a jsou jako freeware volně dostupné na internetu. Ze široké nabídky funkcí, které nabízí, jsou v aplikaci C17 Simulator využity inicializační procedury při započetí komunikace, posílání a čtení bitů z jednotlivých pinů paralelního portu.

Nad těmito zdrojovými kódy pracují unity *GlobDef.pas* a *Common.pas*, které zajišťují propojení low-level komunikace s front-end aplikací, kterou je v tomto případě C17 Simulator. První zmíněná unita *GlobDef.pas* především definuje datové struktury reprezentující paralelní port, signály obvodu C17, jejich směr, zda jsou invertující atd. a několik stavových proměnných. Jejich hodnoty jsou v největší míře nastavovány z okna Nastavení připojení přes paralelní kabel, které je ovládáno unitou Connect.pas nebo jsou z těchto nastavení odvozeny. Dále při inicializaci čte adresy paraleních portů dostupných v PC, na kterém aplikace běží, a zjišťuje verzi operačního systému. Unita *Common.pas* volá metody z výše zmíněné unity *ParaPort.pas* a umožňuje pak posílat a číst jednotlivé bity i celá slova. Zohledňuje při tom například invertující vlastnosti některých pinů nebo to, jak jsou signály obvodu a piny vzájemně propojeny. Používá při tom datové struktury a proměnné z unity *GlobDef.pas*.

Vrchní vrstvou komunikace je samotná aplikace C17 Simulator, která zpracovává požadavky uživatele, jako jsou změny logických vstupů (klikání na tlačítka) nebo změny v nastavení paraleního portu. Ty pak, v příslušném datovém formátu posílá do nižší vrstvy a zároveň z ní čte data příchozí, která vizualizuje pomocí algoritmů z kapitoly 4.4.2.3. Celou strukturu, která není jednoduchá a bylo poměrně složité do ní proniknout jsem se pokusil vystihnout Obrázkem 15.



Obrázek 15: Rozvrstvení komunikace po paralelním portu

5. Závěr

Výsledkem diplomové práce je databáze reprezentativních modelů obvodu C17 obsahující obvody s jednou poruchou. Její součástí je také model obvodu bez poruchy. Originální obvod byl popsán v návrhovém systému ispDesign Expert. Ostatní modely, obsahující poruchy, byly vytvořeny aplikací C17 Bugger. K odladění automatického vkládání poruch bylo taktéž využito prostředí ispDesign Expert.Ke kompilaci návrhu byl použit software ispCompiler a nahrávání hotových modelů bylo prováděno pomocí programu ispDaisy Chain Download přes paralelní port a Lattice Download kabelem do programovatelného hradlového pole ispLSI2032. Výše zmíněný hardware i software, kromě aplikace C17 Bugger, která je jedním z výsledků diplomové práce, je vyráběn firmou Lattice Semiconductor.

Aplikace C17 Bugger nabízí jednoduché prostředí pro vkládání vybraných poruch do VHDL popisu obvodu C17. Omezení této aplikace spočívá v tom, že dokáže pracovat pouze s VHDL popisek, který je součástí instalace aplikace. Návrh uživatelského prostředí byl prováděn s ohledem na přehlednost a zamezení vzniku chyb způsobených uživatelem, protože se uvažuje o zařazení aplikace do výuky.

C17 Simulator je druhou, mnohem rozsáhlejší, aplikací, která vznikla při plnění cílů diplomové práce. Při jejím vzniku byl kladen velký důraz jak na funkčnost tak na vzhled, přičemž jsem čerpal inspiraci z mnoha profesionálních aplikací pro Windows a z literatury. Program procházel postupným vývojem a v době odevzdání diplomové práce se nacházel v dokončené fázi, kdy umí simulovat obvod C17, a to jak ve stavu bez poruchy, tak i za přítomnosti většiny možných poruch a dovoluje tak porovnávání chování s reálným obvodem. Nepodařilo se pouze zpracovat simulaci poruchy ShortCut (zkrat) se zpětnou vazbou přes jedno nebo více hradel, která vede k vzniku obvodů podobných RS klopným obvodům. Přes počáteční problémy se úspěšně podařilo implementovat komunikaci přes paralelní port. Robustnost aplikace byla otestována na několika počítačích s různými verzemi operačního systému od Windows 98 Second Edition až po Windows XP Corporation Edition.

Podrobné manuály k obsluze obou aplikací a návod na práci s modely obvodu jsou součástí diplomové práce.

Ve vývoji poslední zmíněné aplikace by jistě bylo možné ještě dále pokračovat. Nabízí se možnost spouštění automatického testovacího postupu nebo skriptu, který by prošel všechny kombinace pěti logických vstupů a z chování výstupů, případně vnitřních stavů by se identifikovala přítomná chyba. Nebylo by od věci ani implementovat ukládání datových toků mezi aplikací a reálným obvodem do logovacího souboru apod.

Rozšíření by se mohlo týkat i popisu obvodu C17, který by se mohl rozšířit o výstup vnitřních stavů obvodu. V tom případě, co se paralelní komunikace týče, by ale bylo nutné pracovat minimálně s obousměrným módem SP/2 paralelního portu, protože klasické LPT nenabízí dostatek vstupních pinů.

6. Použitá literatura

[1] Cantu, M.: Mistrovství v Delphi 4, Grada Publishing, 1999, ISBN 80-7169-800-8

[2] Kolouch, J.: Programovatelné logické obvody a modelování číslicových systémů v jazycích ABEL a VHDL. Skriptum FEI VUT Brno, 2000, ISBN 80-214-1733-1

[3] Skalický, R.: Vývojová deska pro diagnostické účely, DP TUL 2003

[4] Prof. Ing. Novák, O., CSc.; Prof. Ing. Nouza, J., CSc.; Doc. Ing. Doležal, I., CSc.; Ing. Kolář, M., CSc. Elektronika, Skriptum TU v Liberci, 2001, ISBN 80-7083-499-4

[5] Martinec, T.: Vytvoření řídící aplikace s využitím PC a LPT portu, Časopis K^7 (k7.vslib.cz), číslo 2. ročník 2004, ISSN 1214-7370

[6] Ing. Kolář, M., Csc.: Přednášky z předmětu Elektronická zařízení, KES, TU v Liberci, 2000, dostupné na webové adrese: http://www.fm.vslib.cz/~kes

[7] Dr. Gramatová, E.: Přednášky z předmětu Diagnostika a spoľahlivosť, FIIT STU v Bratislavě, 2000, dostupné na webové adrese http://www.ui.savba.sk/diag/edu.html

[8] Prof. Ing. Novák, O., CSc.: Přednášky z předmětu Diagnostika a spolehlivost, TU v Liberci, 2004, dostupné na webové adrese: http://www.fm.vslib.cz/~kes/pages/dsi04.html

[9] Internetové stránky firmy Lattice Semiconductor Copr.: http://www.latticesemi.com

[10] Internetové stránky firmy SpecoSoft: www.specosoft.com

[11] Dokumentace k obvodu ispLSI2032.

7. Přílohy

7.1. Příloha A – Postup práce s modely

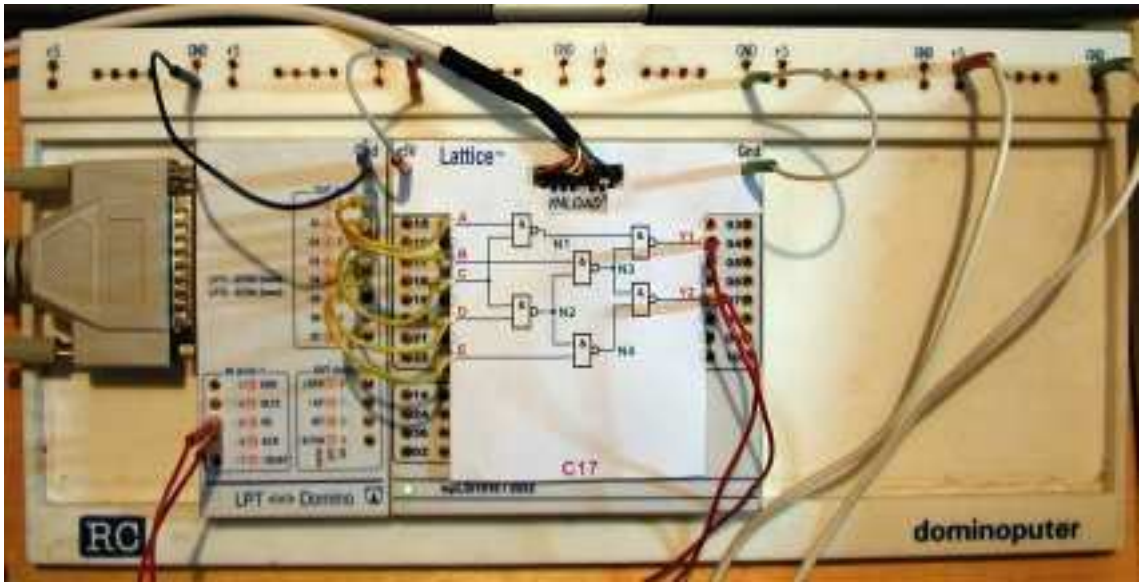
Aplikace C17 Bugger umožňuje vytvořit VHDL popis obvodu C17 s jednou libovolnou poruchou. Pokud chce uživatel model prakticky používat, tzn. nahrát do programovatelného čipu, např. Lattice ispLSI2032, musí provést několik operací, které jsou detailně popsány v této příloze.

7.1.1. Hardware a software

K následující činnosti potřebujeme hardware a software uvedený v seznamu.

a) **Hardware**:
   - Moduly stavebnice Domino:
     - LPT ⇔ Domino
     - ispLSI1016/2032
   - Propojovací vodiče ke stavebnici Domino:
     - 8 krátkých
     - 2 dlouhé
   - Napájecí zdroj Z5 5V/3A = s přívodními vodiči
   - Nahrávací kabel ispDownload Cable
   - Paralelní kabel

b) **Software**:
   - nainstalovaný softwarový balík ispLEVER Starter od firmy Lattice Semiconductor obsahující tyto programy:
     - ispDesignEXPERT
     - ispEXPERT Compiler
     - LSC ISP Daisy Chain Download
   - nainstalovanou aplikaci C17 Bugger a C17 Simulator



obrázek 16: Hardware pro praktickou práci s modely obvodů

Poznámky k zapojení:

1. *Při zapojování je vhodné použít stojan pro moduly Domino, zejména k rozvádění +5 a GND*
2. *Při propojení LPT portu, resp. Domino modulu **LPT ⇔ Domino** je vhodné:*
    - *pro vstupní signály obvodu C17 (a, b, c, d, e) použít výstupní datové piny LPT portu D0 – D7 (piny 2 – 9)*
    - *pro výstupní signály obvodu C17 (y1, y2) použít vstupní piny LPT portu ACK, / BUSY, PE, SLTC nebo ERR na pinech 10, 11, 12, 13, 15*

*Je **nutné** uzemnit piny 18 – 25 paralelního portu (zdířka Gnd na modulu LPT ⇔ Domino), jinak nebude LPT komunikace správně fungovat.*

7.1.2. Od VHDL k JEDEC souboru

1. **Vytvoření projektu:**
    - vytvořit VHLD popis obvodu pomocí aplikace C17 Bugger
    - nakopírovat do prázdného adresáře pro projekt
    - nakopírovat pomocné vhd soubory hradel
    - spustit aplikaci ispDesignEXPERT Project Navigator
    - File → New Project … - vytvořit nový projekt
    - Project Type: Schematic/VHDL
    - uložit do vybraného adresáře s vhd soubory, název libovolný, např. c17
    - Source → Import … - vybrat soubor c17xxxxx.vhd a zvolit VHDL module

- přidat do projektu potřebné soubory hradel Source → Import … vybrat hr_xxx.vhd a zvolit VHDL module
- nastavit čip, který používáme *doubleclick* na čip ispLSI5384VA–125LB388:
  - Family: ispLSI 2k Device
  - Device: ispLSI2032
  - Speed grade: 80 MHZ
  - Package: 44PLCC
  - Op. conditions: Commercial
  - zkontrolovat Part Name: ispLSI2032-80LJ44

2. **Práce s projektem**:
   - můžeme provádět časovou simulaci:
     - Source → New … - vybrat Waveform Stimulus pro ispLSI2032-80LJ44
     - Waveform Editing Tool – nazvat testovací soubor
     - přidat signály – Edit → Import Wave … - vybrat všechny
     - graficky nastavit průběh testovacích vln vstupních signálů
     - uložit File → Save
     - v Project Navigatoru *click* na <soubor>.wdl, lze spustit Functional Simulation nebo Timing Simulation
   - spustit kompilátor – Tools → Compiler Design Environment

3. **ispEXPERT Compiler** - kompilace projektu, vytváření JEDEC souboru
   - Assign Pin Location
   - *Click* Unassigned Pins na signál **a**
   - *Doubleclick* na vybraný pin čipu ispLSI2032 na schématu. Doporučené propojení:

Signál	Pin
a	25
b	27
c	28
d	30
e	32
y1	38
y2	41

Tabulka 9: Doporučené propojení

- Compiler – provede kompilaci modelu a vytvoří JEDEC soubor <název projektu>.jed
- spustit nahrávací software Tools → ispDCD

4. LSC ISP Daisy Chain Download – nahrávací software

- připojit nahrávací kabel
- zapnout napájení obvodu
- New Configuration Setup File- nový konfigurační soubor, vybrat ISP Chain Interface
- Scan Board – automaticky zvolí správný druh používaného čipu, na výzvu není nutné ukládat
- Browse – vybrat JEDEC soubor
- zkontrolovat Program & Verify (PV)
- Run Operation – model obvodu se nahraje do čipu

5. **C17 Simulator**

- Otevřít dialogové okno Nastavení připojení přes paralelní port
- Zvolit používaný LTP port
- Nastavit propojení signálů s piny LPT portu (předdefinované Lattice nebo vlastní)
- připojit LPT kabel k modulu LPT ⇔ Domino
- propojit piny LPT portu s piny čipu (podle nadefinovaného propojení)

Poznámka:

Pro ověření funkce paralelního portu je vhodné použít logické sondy LOG PROBE ze stavebnice Domino.

7.2. Příloha B – Pravdivostní tabulka obvodu C17

a	b	c	d	e	n1	n2	n3	n4	y1	y2
0	0	0	0	0	1	1	1	1	0	0
0	0	0	0	1	1	1	1	0	0	1
0	0	0	1	0	1	1	1	1	0	0
0	0	0	1	1	1	1	1	0	0	1
0	0	1	0	0	1	1	1	1	0	0
0	0	1	0	1	1	1	1	0	0	1
0	0	1	1	0	1	0	1	1	0	0
0	0	1	1	1	1	0	1	1	0	0
0	1	0	0	0	1	1	0	1	1	1
0	1	0	0	1	1	1	0	0	1	1
0	1	0	1	0	1	1	0	1	1	1
0	1	0	1	1	1	1	0	0	1	1
0	1	1	0	0	1	1	0	1	1	1
0	1	1	0	1	1	1	0	0	1	1
0	1	1	1	0	1	0	1	1	0	0
0	1	1	1	1	1	0	1	1	0	0
1	0	0	0	0	1	1	1	1	0	0
1	0	0	0	1	1	1	1	0	0	1
1	0	0	1	0	1	1	1	1	0	0
1	0	0	1	1	1	1	1	0	0	1
1	0	1	0	0	0	1	1	1	1	0
1	0	1	0	1	0	1	1	0	1	1
1	0	1	1	0	0	0	1	1	1	0
1	0	1	1	1	0	0	1	1	1	0
1	1	0	0	0	1	1	0	1	1	1
1	1	0	0	1	1	1	0	0	1	1
1	1	0	1	0	1	1	0	1	1	1
1	1	0	1	1	1	1	0	0	1	1
1	1	1	0	0	0	1	0	1	1	1
1	1	1	0	1	0	1	0	0	1	1
1	1	1	1	0	0	0	1	1	1	0
1	1	1	1	1	0	0	1	1	1	0

Tabulka10: Pravdivostní tabulka obvodu C17

7.3. Příloha C – Analýza problému zkratů se zpětnou vazbou

Analýza problému ShortCut se Zpětnou Vazbou v obvodu C17

Ovlivněné signály: Pokud zkratujeme signály, které se přímo (přes jedno hradlo) i nepřímo (přes více hradel) ovlivňují, tzn. jejich stav je na sobě vzájemně závislý, vzniká sekvenční obvod. Takový obvod nelze simulovat pomocí truth-table - neprovádím v aplikaci C17.

Signály	Ovlivněné signály - vpřed		Ovlivněné signály - vzad	
	přímo	nepřímo	přímo	nepřímo
a	n1	y1		
b	n3	y1, y2		
c	n1, n2	n3, n4, y1, y2		
d	n2	n3, n4, y1, y2		
e	n4	y2		
n1	y1		a, c	
n2	n3, n4	y1, y2	c, d	
n3	y1, y2		b, n2	c, d
n4	y2		n2, e	c, d
y1			n1, n3	a, c, b, n2, d
y2			n3, n4	b, n2, c, d, e

Zakázané ShortCuty Zkraty, které vedou k sekvenčnímu obvodu.

Signál	Zakázaný ShortCut	Setříděné
a	n1 y1	n1, y1
b	n3 y1, y2	n3, y1, y2
c	n1, n2 n3, n4, y1, y2	n1, n2, n3, n4, y1, y2
d	n2 n3, n4, y1, y2	n2, n3, n4, y1, y2
e	n4 y2	n4, y2
n1	y1 a, c	a, c, y1
n2	n3, n4 y1, y2 c, d	c, d, n3, n4, y1, y2
n3	y1, y2 b, n2 c, d	b, c, d, n2, y1, y2
n4	y2 n2, e c, d	c, d, e, n2, y2
y1	n1, n3 a, c, b, n2, d	a, b, c, d, n1, n2, n3
y2	n3, n4 b, n2, c, d, e	b, c, d, e, n2, n3, n4

Tabulka 11: Analýza problému zkratů se zpětnou vazbou

7.4. Příloha D – Seznam obrázků